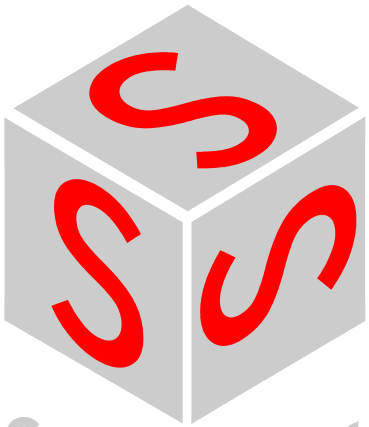
Softmotion

in CoDeSys 2.3

User Manual

This manual is an add-on to the
User Manual for the
CoDeSys Programming System



Copyright © 2003, 2004, 2005 by 3S - Smart Software Solutions GmbH
All rights reserved.

We have gone to great lengths to ensure this documentation is correct and complete. However, since it is not possible to produce an absolutely error-free text, please feel free to send us your hints and suggestions for improving it.

Trademark
Intel is a registered trademark and 80286, 80386, 80486, Pentium are trademarks of Intel Corporation.
Microsoft, MS and MS-DOS are registered trademarks, Windows is a trademark of Microsoft Corporation.

Publisher

3S - Smart Software Solutions GmbH
Memminger Strasse 151
D-87435 Kempten
Tel. +49 831 5 40 31 - 0
Fax +49 831 5 40 31 - 50

Last update: **15.08.2005 (CoDeSys V2.3.5.0)**

Version: **2.5**

Content

1 Softmotion Concept and Components Overview 1-1

2 The SoftMotion Drive Interface 2-1

2.1 PLC Configuration for SoftMotion ... 2-1
2.1.1 BusInterface ... 2-2
2.1.2 AxisGroup ... 2-2
2.1.3 Drive ... 2-4
2.1.4 Encoder ... 2-7
2.2 SM_DriveBasic.lib and automatic Code Generation ... 2-7
2.2.1 Mathematic auxiliary modules of SM_DriveBasic.lib ... 2-7
2.2.2 AxisGroup modules ... 2-8
2.2.3 Configuration Modules ... 2-9
2.2.4 Controller Mode Modules ... 2-10
2.2.5 ControlAxis function blocks ... 2-10
2.2.6 Virtual time axis ... 2-12
2.2.7 Referencing via digital hardware inputs ... 2-12
2.2.8 Diagnosis modules ... 2-14
2.2.9 Encoder ... 2-15
2.2.10 Visualization templates ... 2-15
2.3 Drive Driver <BusInterfaceName>Drive.lib ... 2-15
2.3.1 SercosDrive.lib ... 2-15
2.3.2 SM_CAN.lib ... 2-16
2.4 Variables of the AXIS_REF structure ... 2-16
2.5 Parameterizing of the drive ... 2-20

3 The CNC-Editor in CoDeSys 3-1

3.1 Overview ... 3-1
3.2 Supported and extended elements of the CNC-language DIN66025 ... 3-2
3.3 Start, Inserting and Managing of CNC Programs ... 3-4
3.4 CNC Text editor ... 3-8
3.5 CNC Graphic Editor ... 3-8
3.6 Commands and Options in the CNC-Editor ... 3-8
3.7 Automatic structure filling in the CNC-Editor ... 3-11

4 The CAM-Editor 4-1

4.1 Overview ... 4-1
4.2 Definition of a CAM for SoftMotion ... 4-1
4.3 Starting the CAM-Editor and Inserting a new CAM ... 4-1
4.4 Editing a CAM ... 4-3
4.4.1 General Editor Settings ... 4-3
4.4.2 Editing the properties of a particular CAM element: ... 4-4
4.4.3 Commands of the 'Extras' and 'Insert' Menus ... 4-6
4.5 CAM data structures ... 4-8
4.5.1 Example for a manually created CAM ... 4-10

5 The Library SM_PLCopen.lib 5-1

5.1 Overview ... 5-1
5.2 PLCopen-Specification "Function blocks for motion control, Version 1.0" ... 5-1
5.3 Modules for Controlling Single-Axis Motions ... 5-2
5.4 Modules for Synchronized Motion Control ... 5-15
5.5 Additional Elements of the SM_PLCopen.lib ... 5-19

6 The Library SM_CNC.lib 6-1

6.1 Overview ... 6-1
6.2 Modules ... 6-1
6.2.1 SMC_NCDecoder ... 6-1
6.2.2 SMC_ToolCorr ... 6-3
6.2.3 SMC_AvoidLoop ... 6-5
6.2.4 SMC_SmoothPath ... 6-6
6.2.5 SMC_RoundPath ... 6-8
6.2.6 SMC_CheckVelocities ... 6-9
6.2.7 SMC_Interpolator ... 6-11
6.2.8 SMC_Interpolator2Dir ... 6-15
6.3 Auxiliary Modules for Path Rotations, Translations and Scalings ... 6-16
6.4 Settings via global variables ... 6-17
6.5 Structures in the SM_CNC.lib ... 6-17

7 The library SM_CNCDiagnostic.lib 7-1

7.1 Function blocks for the analysis of SMC_CNC_REF data ... 7-1
7.1.1 The function block SMC_ShowCNCREF ... 7-1
7.2 Function blocks for analysis of SMC_OutQueue data ... 7-1
7.2.1 The function block SMC_ShowQueue ... 7-1

8 The Library SM_Trafo.lib 8-3

8.1 Overview ... 8-3
8.2 Transformation function blocks ... 8-3
8.2.1 Portal Systems ... 8-3
8.2.2 Portal Systems with Tool Offset ... 8-4
8.2.3 H-Portal-System with stationary drives ... 8-8
8.2.4 2-Jointed Scara-Systems ... 8-9
8.2.5 3-Jointed Scara-Systems ... 8-11
8.2.6 Parallel Kinematics ... 8-13
8.3 Spacial Transformations ... 8-15

9 The Library SM_Error.lib 9-1

9.1 Function blocks ... 9-1
9.1.1 SMC_ErrorString ... 9-1
9.2 The enumeration SMC_Error ... 9-1

10 The library SM_FileFBs.lib 10-1

10.1 Overview ... 10-1
10.2 CNC function blocks ... 10-1
10.3 CAM Function Blocks ... 10-3
10.4 Diagnosis function blocks ... 10-3

11 Programming Examples 11-1

11.1 Overview ... 11-1

11.2 Example: Drive Interface: Create PLC Configuration for Drives 11-1
11.3 Example: Single Axis Motion Control 11-4
11.4 Example: Single-Axis Motion Control in CFC with Visualization-Template 11-5
11.5 Drive Control via CAM and a Virtual Time Axis 11-7
11.6 Example: Changing CAMs 11-8
11.7 Example: Drive Control via the CNC-Editor 11-8
11.7.1 CNC Example 1: Direct Creation of the OutQueue 11-8
11.7.2 CNC Example 2: Decoding online with use of variables 11-11
11.7.3 CNC Example 3: Path-Preprocessing online 11-13
11.8 Dynamic SoftMotion-Programming 11-15

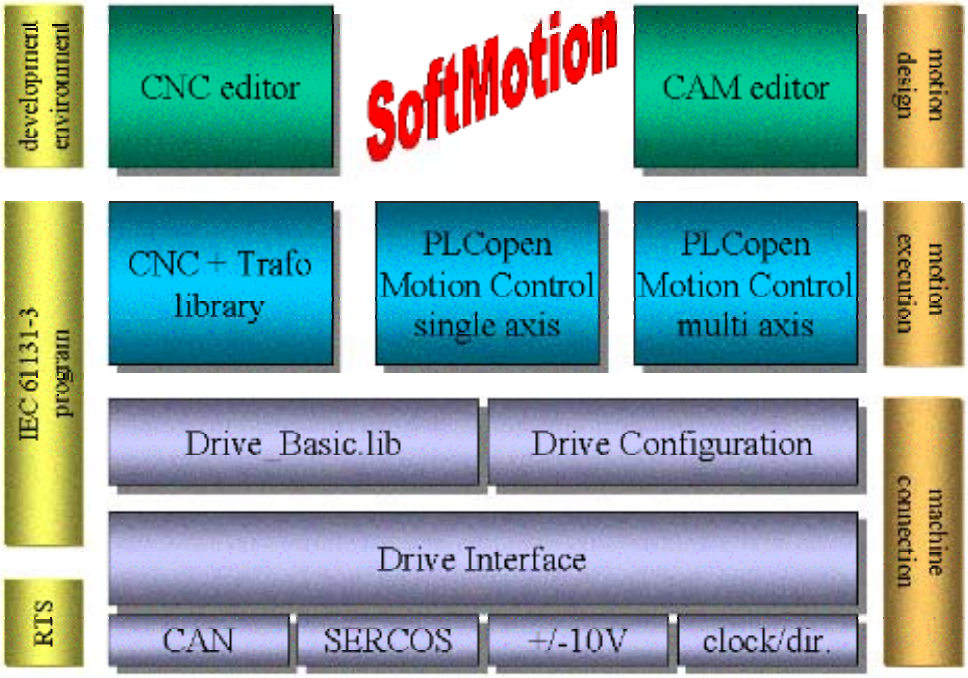
12 Index I

1 Softmotion Concept and Components Overview

SoftMotion allows to realize movements – simple single-axis movements and CAMs as well as complex motions in more dimensions – in the development environment of *CoDeSys*. Particularly applications, where not solely the motion functionality, but also sequence and process control or auxiliary functions are the main thing of the application, are an ideal implementation area for *SoftMotion*.

SoftMotion is a kind of toolkit suitable to exert influence also during the runtime without demanding big effort and detailed know-how for the realization of the desired motions.

The complete program logic is handled in the PLC program and just the pure motion information is executed by the library functions.



SoftMotion can be divided in the following components:

- **Drive Interface**
  This component is responsible for the communication with the drives. It consists of the library *Drive_Basic.lib* and drive- and bussystem-specific libraries and drivers.

- In the **Configuration editor** in CoDeSys the developer maps the structure and configuration of the drive-hardware. Basing on this CoDeSys – using the functions of the **Drive Interface libraries** - will create IEC data structures, which represent the drives abstractedly. Automatically, i.e. without additional effort by the IEC-programmer, the Drive Interface will communicate with the drives and by that will take care of the topicality of the drive data structures as well as of the transfer of the data which have been updated. Based on this structures the drive-controlling IEC program either works with the aid of standard modules of the SoftMotion libraries (SM_CNC.lib, SM_PLCOpen.lib) or with special modules created by the IEC programmer for this purpose.
  The target value always is set cyclically, that means per each IEC task cycle target values (positions, velocities, accelerations etc.) are calculated and transferred from the Drive Interface to the drives. The possibility to "instruct" the drives, like setting a target position so that the drive is moving on its own initiative and giving a message as soon as the instruction has been executed successfully, is not provided. Reasons: In this case no coordinated movements of several axes would be possible and the central controller did not have any influence on the drives during executing an instruction.

- **CNC-Editor**
  The CNC-Editor in CoDeSys allows to program multidimensional motions, which can be transferred and controlled via the drive interface which does the communication with the drive-hardware. The editor works abutted to the CNC language DIN66025, synchronously in a graphical and a text editor. Basically up to 9-dimensioned motions can be realized, whereby only two dimensions will be interpolated not linearly. Thus in two dimensions lines, circles, parables, ellipses and splines can be programmed, the other directions are interpolated just linearly. For each path, which has been designed, CoDeSys automatically creates a data structure, which is available in the IEC program.

- **CAM-Editor**
  The CAM editor, which is integrated in the programming interface of CoDeSys and which is usable graphically, serves to program CAMs for the controlling of multi-axes drives. CoDeSys implicitly creates a global data structure for each programmed CAM. This structures then can be used by the IEC program.

- **CNC-Libraries**
  The “library "SM_CNC.lib", "SM_CNCDiagnostic.lib“and "SM_Trafo.lib" provide modules which can be used to realize, display and execute the motions which have been programmed in the CNC-Editor, resp. which are created during runtime.

- **PLCopen-Library**
  The PLCopen motion control library "SM_PLCopen.lib" contains among other modules which help to program and realize easily as well the controlling of a single axis motion but also of the synchronized motion of two axes. Besides library elements which can be used for status checks, for the parametrizing and for operating in general, there are also function blocks, which can move an axis - according to the defined speed and acceleration parameters – in different ways. If two axes should be synchronized, then one axis serves as master and controls a second axis (slave) according to a certain prescript. This prescript e.g. can be a CAM which has been designed in the CAM editor and which uses available POUs to link the slave axis to the master axis. Further on there are function blocks, which allow electronic gear or phase shifts.

- **File Service Library**
  The library "SM_FileFBs.lib" bases on the system library "SysLibFile.lib" and therefore can only be used on target systems which are supporting this library.

- **Error Library**
  The library "SM_Error.lib" contains all error outputs which can be produced by the modules of the other libraries. Further on it can be used to create German and English error messages from the numeric error variables.

Portability

Apart from some drivers of the Drive Interface, which are serving hardware components directly, all SoftMotion runtime components are programmed in IEC1131-3. Thus the maximum platform independency is reached.

For a quick understanding of the SoftMotion components it is recommended to study the corresponding examples.

2 The SoftMotion Drive Interface

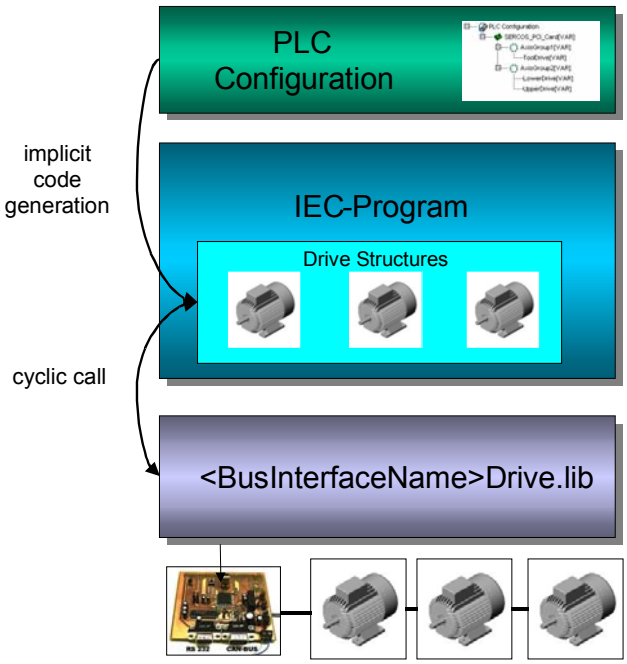
The *Softmotion Drive Interface* is a standardized interface, which allows to include the abstracted image of a drive hardware in the IEC program, and to configure and address it there. It takes care of update and transfer of the motion data which are necessary for controlling the drive hardware. This not only allows easy changing of drives and reuse of IEC programs, but also saves the difficulties and inconveniences of connecting the drives.

The Drive Interface uses the following components:

- The CoDeSys PLC Configuration: Here – basing on a corresponding configuration file - the structure of the drives, which should be controlled, must be mapped by the programmer and the appropriate parameters have to be set. This structure then will be made accessible for the application with the aid of the Drive Interface libraries by implicitly created and assigned (system) variables.

- The internal library Drive_Basic.lib: provides IEC data structures and global variables, which will represent the drives, axisgroups and bus interfaces which have been configured in the PLC Configurator.

- The driver, i.e. the hardware and bus system specific library <BusInterfaceName>Drive.lib (e.g. SercosDrive.lib), which has to be provided by the drive manufacturer offers special functions for the data exchange between the structures and the hardware (see 2.4).



2.1 PLC Configuration for SoftMotion

(Have a look to the programming examples in Chapter 11)

The CoDeSys PLC Configuration usually provides the following elements which can be used to map the structure of the drive hardware:

BusInterface: Field bus interface used for the communication with the drives.

AxisGroup: a physically linked group of drives

Drive: Drive

Encoder: Encoder

The bus interfaces, axisgroups and drives can get any desired but unique IEC 61131-3 identifiers:

Each of these configuration objects can be configured in dialogs if those dialogs are supported by the target. Besides the comfortable configuration via dialogs the parameters can also be set in a configuration list („Module parameters“). There you additionally might find target-specific parameters, preceded by „MS.“.

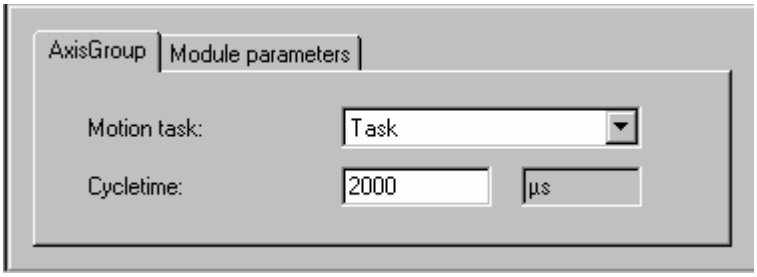
2.1.1 BusInterface

Per default here only the communication driver is selected.

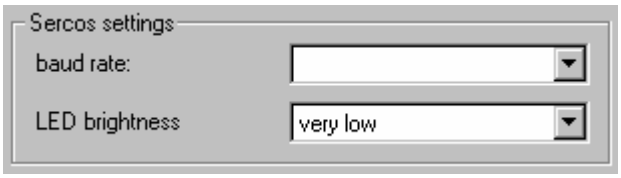
2.1.2 AxisGroup

Here you set the task which controls the communication with the drives, and – if it is not a cyclic task but a event controlled task – its cycle time.

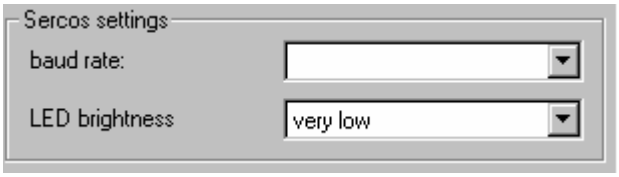
For systems without task configuration this field remains empty.



For **Sercos** interfaces there are further specific settings: the baud rate and the brightness of the LED.



Also for **CAN** axisgroups there are specific settings:



Besides the **baud rate** the number of the used CAN controllers is defined via the **Controller no**. (regard that – if you are using a PLC with two CAN channels and additionally the library 3S-CANopen.lib – this will automatically use Controller 0, and therefore you must select channel 1 for the drives).

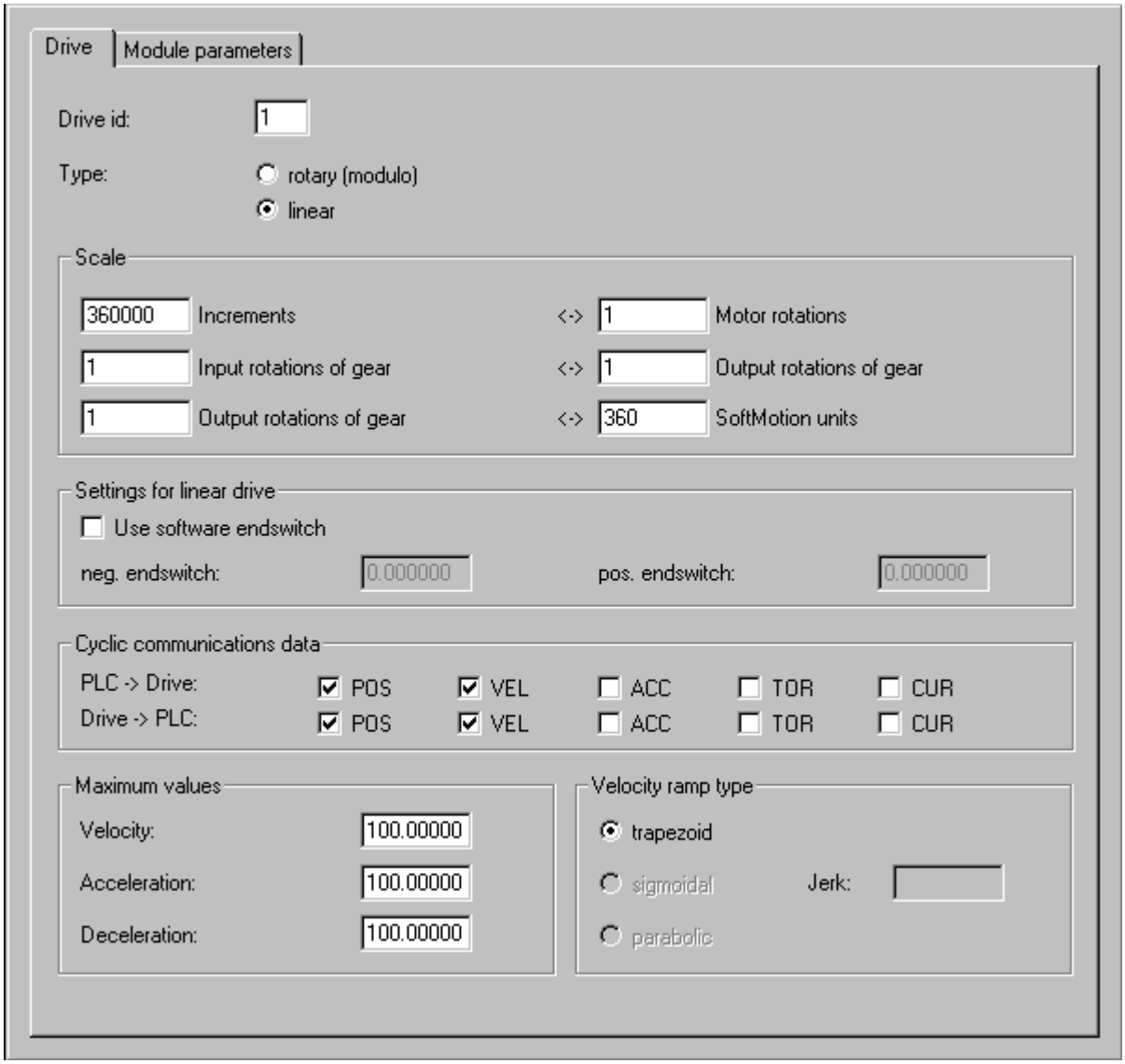
For **SYNC producer**s you can choose between three methods for the synchronization of the drives and the PLC:

- PLC: The PLC is acting as synchronization master. As a rule the user defines the motion task to be a cyclic task. This task calls the driver which at once will send a SYNC-telegram. This method is the simplest, however can lead to problems when used with controllers with high jitter and with drives requesting high accuracy of the SYNC telegram.
- 1.Drive: The first drive (if supporting this feature) creates the SYNC-telegram. The motion task in the PLC then as a rule is defined on the event <AxisGroup>.bSync and thus will wait until a SYNC-telegram has been received and before starting the task processing.
- SYNC device: This method is used if the upper two are not possible. An additional device with CAN ID 127 will be installed in the bus In den Bus, being able to create time-accurate SYNC-telegrams (Index: 1005h, Bit30).

All these settings also can be viewed and modified in the „Module parameters“ dialog.

sTask	String, complying with the name of the task, which will handle the data transfer of this axisgroup
dwCycle	Cycle time (in microseconds) of the task which is defined in "sTask" (only to be defined if the controller does not support tasks and automatically is calling PLC_PRG (Default task))
wParam1 ... wParam4	Card-/Drive specific parameter of type WORD
dwParam1 .. dwParam4	Card-/Drive specific parameter of type DWORD

2.1.3 Drive



In this dialog the **Drive id** is defined. Further on the drive **Type** is selected: linear or rotary (modulo).

Concerning the **Scale** you have to define the conversion between the integer position values and the technical units used in the IEC program. Thereby additionally a drive can be regarded. In the figure shown above, a drive creating 3600000 increments for one rotation would be scaled in a way that the technical units are in radians.

In the **Settings for linear drive**s software endswitches can be defined, **for rotatory drive**s the modulo range must be defined.

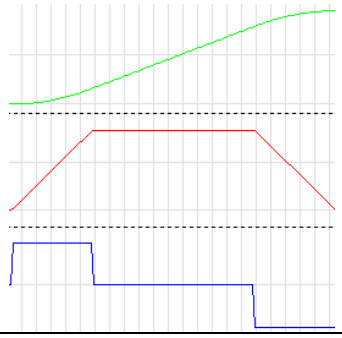
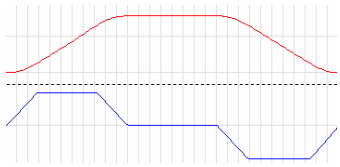
In the **Cyclic communications data** sector define which scheduled resp. actual data should be cyclically transferred to the drive.

In the **Maximum values** sector set the limits which are used by SMC_ControlBy modules in order to detect jumps (see chapter 2.2.4, SMC_ControlAxisByPos).

In the **Velocity ramp type** (if supported by the included libraries) define the velocity profile type for the velocity-generating one-axis and master/slave-modules. “trapezoid“ results in a trapezoid velocity profile (constant acceleration in each section), “sigmoid“ results in a sin²-velocity profile (continuous acceleration), “parabolic“ in a continuous trapezoid and thus parabolic acceleration profile.

For the modes “sigmoidal” and “parabolic” additionally the **Jerk** must be defined.

The following images show which effect the different ramp types have on a positioning. The position is displayed green, the velocity red and the acceleration blue.

Trapezoid ramp mode: There are jumps in the acceleration.	
Sigmoidal ramp mode: The jumps are eliminated. The course of the motion is clearly defined and due to this reason the **jerk** cannot be limited. The definition of the jerk is only used if the drive already at start has an acceleration unequal 0. In this case the acceleration will be jerk-limited run to zero, before the actual movement will be started. Compared to the trapezoid velocity profile this move will last longer.	
Parabolic ramp mode: The acceleration will have a continuous, trapezoid profile, whose gradient will be limited by the jerk. The velocity will have a continuous parabolic profile. Only at this profile the **jerk** actually can be limited.	

All these settings can also be viewed and modified in the „Module parameters“ dialog.

wId	ID of the drive in the axisgroup (WORD)
wControlType	predefined control and return message types (WORD): (<Send data>-> <Return data>) 1. TOR -> --- (Torque -> ---) 2. VEL -> VEL Velocity -> Velocity 3. VEL -> POS Velocity -> Position 4. POS -> POS Position -> Position 5. POS, VEL ->POS, VEL Position -> Velocity 6. VEL -> --- Velocity -> --- 7. CONFIGURABLE manual configuration via wCyclicDataS1, ..S2, ..S3 and wCyclicDataR1, ..R2, ..R3 (see below)

wCyclicDataS1 wCyclicDataS2 wCyclicDataS2 wCyclicDataR1 wCyclicDataR2 wCyclicDataR3	Definition of the send (..S..)- and return data (..R..), if *wControlType* is defined as 'CONFIGURABLE'; options depending on the drive-driver; basically possible: Act/SetPosition Defines the position Act/SetVelocity Defines the velocity Act/SetTorque Defines the torque Act/SetCurrent Defines the current Act/SetUserDef user-defined
dwRatioTechUnitsDenom iRatioTechUnitsNum	Denominator and numerator for the conversion factor at the conversion of bus data to technical units [u]; (DWORD resp. INT)
iMovementType	two options for the motion type: linear / rotary
fPositionPeriod	Period for rotatory axes; depends on the conversion factors 'dwRatioTechUnitsDenom' and 'iRatioTechUnitsNum' (see above)
fSWMaxVelocity	Maximum velocity for software check
fSWMaxAcceleration	Maximum acceleration for software check
fSWMaxDeceleration	Maximum deceleration for software check
bSWLimitEnable	Switching on a software position check (only linear drives), which effects, that the axis will be set to error status as soon as leaving the permissible position range.
fSWLimitNegative	negative position limit (only linear drives)
fSWLimitPositive	positive position limit (only linear drives)
bHWLimitEnable	Switching on a hardware position check (only linear drives), which effects, that the axis will be set to error status as soon as leaving the permissible position range.

For **Sercos** drives a separate dialog is available:

For defining the **Device type** you can choose between “Drive” and “I/O-Device”, because there is no standard CoDeSys-support for Sercos I/Os. If you select “I/O-Device“, some parameters usually transferred by the master will be left out.

Additional cyclic communication data (besides the defaults POS, VEL, ACC, TOR, CUR) can be transferred. For this purpose you must enter the Sercos parameter number (IDN)and length.

The entries in **PackProfile check** are used to check whether the settings have been done according to the PackProfile standard. It will be differentiated between the profiles BasicA, BasicB and Extended. Regard that this check can be performed on the offline-data. Use of the additional configuration mechanism (Read ASCII-file on PLC), which is available for Sercos, might change the result. For this reason in the SercosDrive.lib an additional online-check is implemented (see documentation Sercosdrive.pdf). Besides this together with Sercosdrive.lib XML-files are provided which can be imported in the dialog and which contain all permissible PackProfile parameters.

In the **Init data** section parameters can be defined to be written to the drive during startup. For this purpose there are lists of parameters to be written to the drive in Phase2 resp. Phase3 resp. at start of Phase4. Using the appropriate parameter list you can reach a complete initialization of the drive at the start of the application; which might be useful, if e.g. the drive had to be exchanged (see also chapter 2.5)

Via entries in list „**locked**“ you can avoid an automatic transfer of single parameters by the driver.

All those settings can be saved in a xml-file (button „**save**“) resp. can be read from a xml-file (button „**load**“).

For **CAN**-drives also a special dialog is available where parameters are entered which are to be written to the drive during startup. These also can be stored and reloaded in/from xml-files.

2.1.4 Encoder

wEncoderId	ID of the Encoder (WORD)
dwRatioTechUnitsDenom iRatioTechUnitsNum	Denominator and numerator for the conversion factor of bus data (drive increments) to technical units (units used in application, Softmotion units) [u]; (DWORD resp.. INT)
iMovementType	Encoder type; Selection options: linear or rotary
fPositionPeriod	Period for rotatory axes; depends on the conversion factors 'dwRatioTechUnitsDenom' and 'iRatioTechUnitsNum'
bSWLimitEnable	Enable software position check (only linear drives), which effects, that later the axis connected via SMC_Encoder will be set to error state as soon as it leaves the position window.
fSWLimitNegative	negative position limit (only linear encoders)
fSWLimitPositive	positive position limit (only linear encoders)

2.2 SM_DriveBasic.lib and automatic Code Generation

If the library **SM_DriveBasic.lib** is included in the IEC1131 application in CoDeSys, CoDeSys will automatically generate structure objects based on the drive image which has been configured in the PLC Configuration editor. These structures can be accessed by the IEC program.

Besides that, to the IEC1131 application there **must** be linked a manufacturer specific library fitting to the used hardware. This library must have the name **<BusInterface_name>Drive.lib**. It supports the hardware-specific Drive Interface functionality. The "BusInterface_name" results from the setting which has been made in the PLC Configuration in the module parameters of the bus interface (see entry 'Interface Type'). From that string the left part before the first space is used (Example: "CAN (Peak)" -> "CAN" -> the manufacturer specific library will be named "CANDrive.lib").

During start of the application the implicit call of the functions <BusInterfaceBezeichnung>DriveExecute_Start and <BusInterfaceBezeichnung>DriveInit at the begin of the task and <BusInterfaceBezeichnung>DriveExecute_End at the end of the task will cause the transmission and maintaining of the AXIS_REF-structure variables. In case of errors during initialization of the drives the global variable g_strBootupError contains a error description, which is created by the library <BusInterfaceBezeichnung>drive.lib.

Additionally to its main function, the representation of the drives in the IEC program, the library SM_DriveBasic.lib also contains some auxiliary modules:

2.2.1 Mathematic auxiliary modules of SM_DriveBasic.lib

The function **SMC_sgn** returns the value of the sign of the input; thus –1 if the input is negative, +1 if it is positive and 0 if it is zero.

The function **SMC_fmod** calculates the modulo value of the input x for period m. The return value always is within the interval [0, m].

The function **SMC_atan2** calculates and returns the angle, which solves the following equations:

sin(alpha) * f = Sinus und cos(alpha) * f = Cosinus.

In contrast to the common ATAN function the value range in this case covers the complete interval [0; 2pi].

2.2.2 AxisGroup modules

SMC_IsAxisGroupReady

This function by a boolean variable returns whether the startup, which implicitly is done for each axisgroup during the start or the program, has been terminated and thus the group with its axes is operable (TRUE), or whether the startup is still going on or an error has occurred (FALSE).

SMC_GetAxisGroupState

This function block tells about the status of an axisgroup:

Inputs (VAR_INPUT) of the function block:

bEnable : BOOL

If this entry is TRUE, the module provides information on the status of an axisgroup.

In-/Outputs (VAR_IN_OUT) of the function block:

AxisGroup : SMC_AXISGROUP_REF

Axisgroup, for which information is needed.

Outputs (VAR_OUTPUT) of the function block:

bDone : BOOL

TRUE, as soon as there are valid data on the ouputs.

wState : WORD

Internal state variable of the axis.

bStartingUp : BOOL

Axisgroup startup, i.e. the drives get configured. (0 <= wState <= 99)

bNormalOperation: BOOL

Axisgroup in normal operation. (wState = 100)

bResetting: BOOL

Axisgroup just getting reinitialized. (200 <= wState <= 210)

bErrorDuringStartUp: BOOL

During startup an error occurred. (wState >= 1000)

pErrorDrive: POINTER TO AXIS_REF

Pointer on the error-causing axis. Only valid if bErrorDuringStartUp = TRUE. With the aid of this output the erroneous axis can be removed from the axisgroup during runtime by setting the variable DisableDriveInAxisGroup. Then the axis can be reinitialized and the drive can be continued with the remaining axis, if there are redundancies available in the machine.

SMC_ResetAxisGroup

With this function block a complete axisgroup can be reinitialized.

Inputs (VAR_INPUT) of the function block:

bExecute : BOOL

If this input is TRUE, the module starts reinitializing the axisgroup.

bKeepRatioSettings: BOOL

If this input is TRUE, the recent drive settings (dwRatioTechunitsDenom and iRatioTechUnitsNum), the modulo value (fPositionPeriod) and the axis type (iMovementType, linear/rotatory) will be kept and not be replaced by the values which are set in the PLC Configuration.

In-/Outputs (VAR_IN_OUT) of the function block:

AxisGroup : SMC_AXISGROUP_REF

Axisgroup to be reinitialized.

Outputs (VAR_OUTPUT) of the function block:

bDone : BOOL

TRUE, if process is terminated.

bError : BOOL

Error occured.

nErrorID: SMC_ERROR

Error description.

2.2.3 Configuration Modules

SMC_ChangeGearingRatio

With the aid of this module the IEC program can change the gearing ratio and the type of the drive.

After execution of this module the axisgroup should be restarted by SMC_ResetAxisGroup (bKeepRatioSettings=TRUE), in order to guarantee that all variables are initialized correctly!

Inputs (VAR_INPUT) of the module:

bExecute : BOOL

At a rising edge the module will start.

dwRatioTechUnitsDenom : DWORD

iRatioTechUnitsNum: DWORD

SoftMotionUnit–increments conversion ratio of (see 2.1).

fPositionPeriod: LREAL

Position period, modulo value (only for rotatory drives) (see 2.1).

iMovementType: INT

o: rotatory axis, 1: linear axis.

In-/Outputs (VAR_IN_OUT) of the module:

Axis : AXIS_REF

Drive for which the gearing ratio should be changed.

Outputs (VAR_OUTPUT) of the module:

bDone : BOOL

TRUE, as soon as the action has been executed.

bError : BOOL

TRUE, if error occurred.

nErrorID : SMC_Error

Error description.

2.2.4 Controller Mode Modules

SMC_SetControllerMode

If supported by the drive this module can be used to switch to another controller mode.

Inputs (VAR_INPUT) of the module:

bExecute : BOOL

Mit einer steigenden Flanke wird der Baustein aktiv.

nControllerMode: SMC_CONTROLLER_MODE

Desired controller mode: SMC_torque (torque), SMC_velocity (velocity), SMC_position (position), SMC_current (current)

In-/Outputs (VAR_IN_OUT) of the module:

Axis : AXIS_REF (VAR_IN_OUT)

Drive for which the controller mode should be changed.

Outputs (VAR_OUTPUT) of the module:

bDone : BOOL (VAR_OUTPUT)

TRUE, as soon as action has been executed.

bError : BOOL (VAR_OUTPUT)

TRUE, if error occurred.

nErrorID : SMC_Error (VAR_OUTPUT)

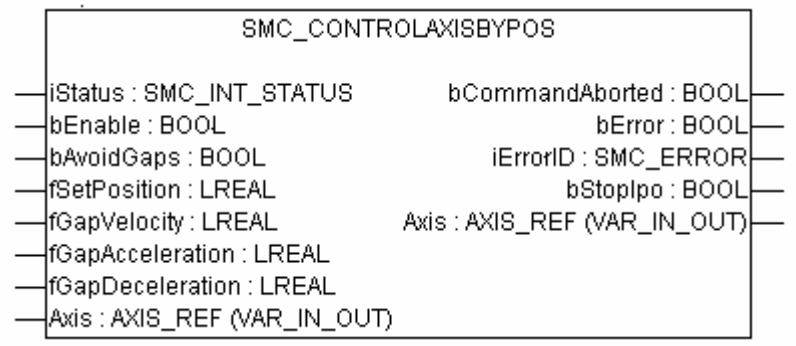
Error description.

2.2.5 ControlAxis function blocks

These modules can be used to control a drive by direct setting of the desired values:

SMC_ControlAxisByPos

This function block writes target positions n to a drive structure and checks the structure for jumps.



Inputs of the function block:

iStatus: SMC_INT_STATUS

State of the Interpolation module. Gets connected with the homonymous output of SMC_Interpolator.

bEnable: BOOL

Controls the axis as long as is TRUE.

bAvoidGaps: BOOL

TRUE: The module watches position and velocity. If the velocity exceeds the limit fSWMaxVelocity, which is stored in the axis (configured in the drive dialog in „Maximum values“), then the module will set output bStopIpo and move the axis according to the parameters fGapVelocity, fGapAcceleration and fGapDeceleration to this position and then will delete output bStopIpo.

fSetPosition: LREAL

Target position of the axis. Typically this is an output of the Transformation module.

fGapVelocity, fGapAcceleration, fGapDeceleration: LREAL

Move parameters for bridging a jump.

Outputs of the function block:

bCommandAborted : BOOL (Default: FALSE)

TRUE: The module has been aborted by another one.

bError : BOOL (Default: FALSE)

TRUE: An error has occurred in the module.

iErrorID : SMC_Error (INT)

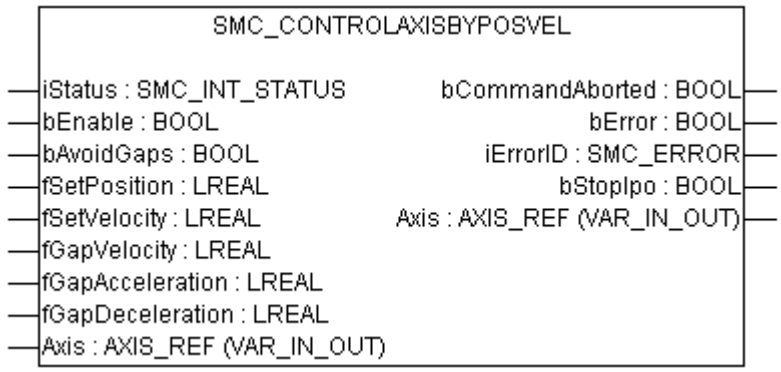
Error number

bStopIpo : BOOL (Default: FALSE)

TRUE: the module has detected a jump in velocity or position and is just adjusting to the new position. For this reason this output should be connected to the EmergencyStop-input of the SMC_Interpolator, so that the Interpolator will wait until the axis is positioned correctly.

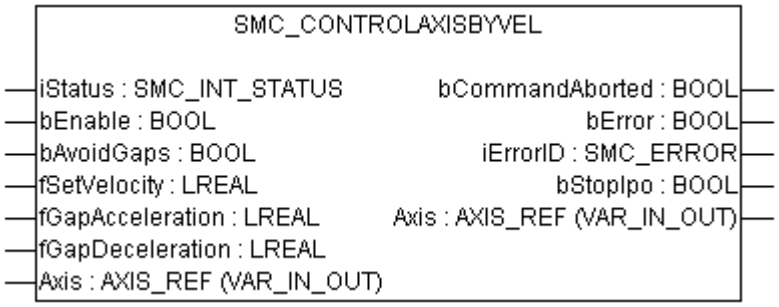
SMC_ControlAxisByPosVel

This module works similar to SMC_ControlAxisByPos, but additionally the velocity can be defined.

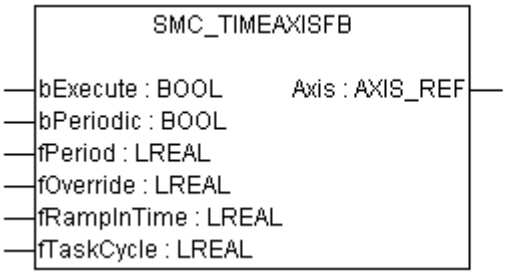


SMC_ControlAxisByVel

This module works similar to SMC_ControlAxisByPos, but the axis is not controlled by the position but by the velocity.



2.2.6 Virtual time axis



This function block creates a time axis, which will be given out by the output *Axis* (AXIS_REF).

At a rising edge in input *bExecute* the target position of the time axis starts to count up in seconds, starting with 0. If input *bPeriodic* is set, then it will restart with 0 as soon as the time given by *fPeriod* has been reached.

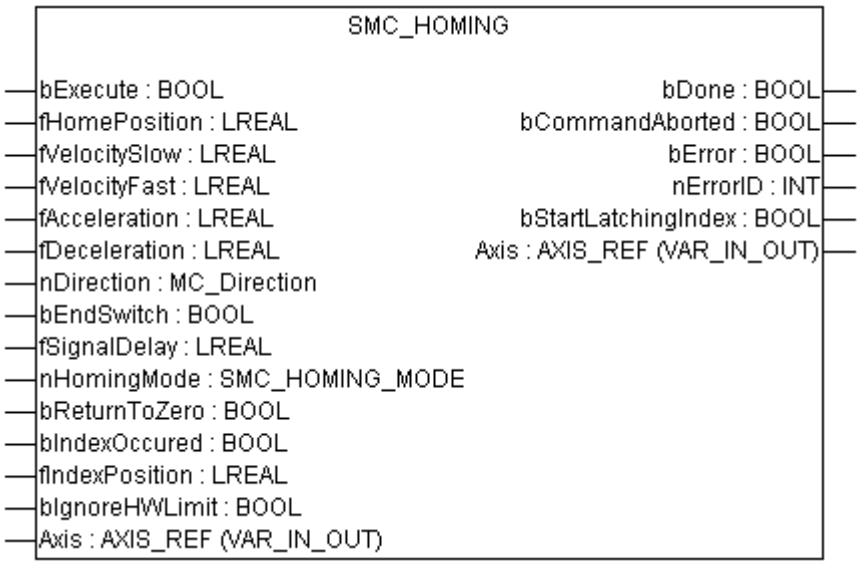
Input *fOverride* gives a time multiplicator, which per default is set to 1. A "2" would make the time running twice as fast..

Input *fRampInTime* defines how long the function block can take to ramp on the new override after the new target values have been read.

In input *TaskCycle* define the cycle time (seconds) of that task which is calling the function block.

2.2.7 Referencing via digital hardware inputs

SMC_Homing



This function block can execute the reference move of an axis. As an ON-switch a boolean value is used, typically a hardware input.

After the module has been started with a rising edge in *bExecute*, it moves the axis at a velocity *fVelocityFast* in a direction defined by *nDirection*, until the *bEndSwitch* = FALSE, i.e. the reference switch, will be closed. Then the axis will be slowed down and driven in the opposite direction according to *fVelocitySlow*. The reference position will be set and the drive will be stopped at exactly that point where the reference switch opens (*bEndSwitch* = TRUE).

Inputs of the module:

bExecute : BOOL (Default: FALSE)

At a rising edge the reference motion of the drive will be started.

fHomePosition : REAL

Absolute position on the reference position [u].

fVelocitySlow, fVelocityFast : REAL

Target velocity for phase 1 and 2 in [u/s].

fAcceleration, fDeceleration : REAL

Target acceleration and deceleration in [u/s²].

nDirection : MC_Direction (Default: negative)

Direction of the reference motion: permissible values: positive/negative.

bEndSwitch : BOOL (Default: TRUE)

Reference switch: TRUE (open), FALSE (closed).

fSignalDelay : REAL (Default: 0.0)

Transmission time of the reference switch in s. If a time >0 is set, the module will not use the position at which the bEndSwitch has got TRUE as a reference position, but that position which the axis had *fSignalDelay* seconds before.

nHomingMode : SMC_HOMING_MODE (Default: FAST_BSLOW_S_STOP)

FAST_BSLOW_S_STOP:

The drive will be moved to the given direction at velocity *fVelocityFast* (FAST) until the input *bEndSwicth* gets FALSE, then will be stopped and moved to the opposite direction at velocity *fVelocitySlow* (BSLOW) until *bEndSwitch* gets TRUE again. At this position the reference point will be set (S) and it will be stopped (STOP).

FAST_BSLOW_STOP_S:

In contrast to FAST_BSLOW_S_STOP after the free move first a stop is done and afterwards the reference point is set.

FAST_BSLOW_I_S_STOP:

In contrast to FAST_BSLOW_S_STOP after the first free move an index impulse (bIndexOccured=TRUE) and its position fIndexPosition, set as reference point, will be awaited. Not until then it will be stopped.

FAST_BSLOW_S_STOP/ **FAST_BSLOW_STOP_S** / **FAST_BSLOW_I_S_STOP:**

These modes work exactly like those described above except that there will not be turned reverse when having reached the reference switch but will be moved on Regard that in this modes input bIgnoreHWLimits for safety reasons must be FALSE.

bReturnToZero: BOOL (Default: FALSE)

If this flag is set, the module will set the position on the zero point after having terminated the procedure which is defined by *nHomingMode*.

bIndexOccured: BOOL (Default: FALSE)

Only for nHomingMode FAST_BSLOW_I_S_STOP: Indicates whether the index pulse has occurred.

fIndexPosition: REAL (Default: 0.0)

Only for nHomingMode FAST_BSLOW_I_S_STOP: Latched position of the index pulse. If this entry is TRUE, the hardware control of the end switches will be intermitted. Choose this option if you use the same physical switch as hardware-end and reference switch.

bIgnoreHWLimit: BOOL (Default: FALSE)

As long as this input is TRUE, the hardware control of the end switches will not be done. Use this option, if you are using the same physical switch for the hardware end switch and the reference switch.

Outputs of the module:

bDone : BOOL (Default: FALSE)

If TRUE, the drive is referenced and in standstill.

bCommandAborted : BOOL (Default: FALSE)

If TRUE, the command has been aborted by another one.

bError : BOOL (Default: FALSE)

TRUE indicates an function block error.

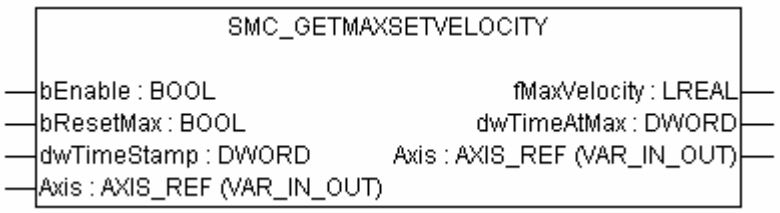
nErrorID : SMC_Error

Error number.

2.2.8 Diagnosis modules

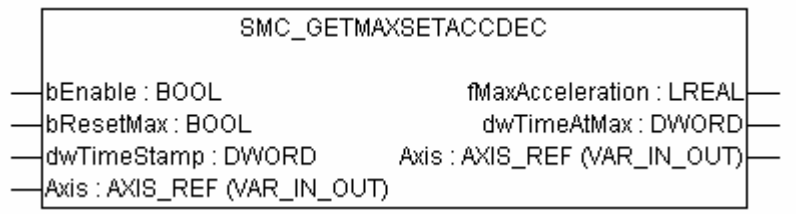
SMC_GetMaxSetVelocity

This function block can measure the value of the maximum (target) velocity of an axis. The measuring will be done if *bEnable* is TRUE, and it will be set back to 0, as long as *bResetMax* is TRUE. With *dwTimeStamp* you can read any DWORD (e.g. call counter), which is taken over and output with a new maximum value.



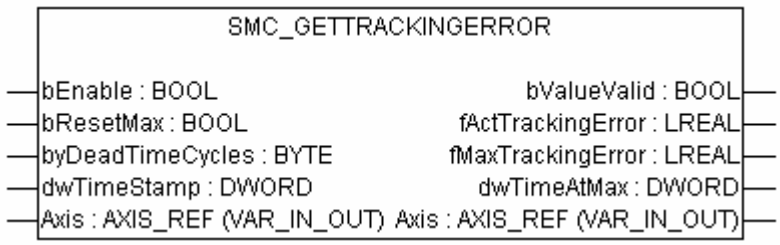
SMC_GetMaxSetAccDec

This function block works analogically to SMC_GetMaxSetVelocity and determines the acceleration or deceleration value which according to amount is the highest.



SMC_GetTrackingError

This function block measures the actual and maximal lag error again the dead time, which can arise from the communication via a field bus and which is given in number of cycles (*byDeadTimeCycles*). Like with SMC_GetMaxSetVelocity a time stamp (*dwTimeStamp*) can be used to measure the time at the maximum.



2.2.9 Encoder

Using the PLC Configuration you can add encoders to an axisgroup and cofigure them. The data structures if type SMC_ENCODER_REF must be processed by an instance of the SMC_Encoder module. This instance will provide as an output an AXIS_REF data structure, which - as soon as the output bValid has confirmed the validity of the data – will serve as an input for all other function blocks (e.g. MC_CamIn, MC_GearIn, MC_TouchProbe).

Via the boolean input bSet the current value of the encoder can be set on the input fSetValue.

2.2.10 Visualization templates

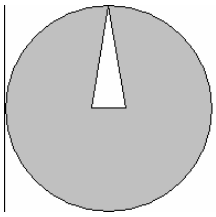
For each of the two drive types linear / rotatory) the library contains a visualization template, which can be linked to the axis structure (AXIS_REF) in order to visualize the current position of the drive:

LinDrive

For a linear drive this picture will be displayed. The slide will be positioned according to its current position relating to the lower and upper position limit and gets blue as soon as it is in regulation. It is a precondition for the use of the template, that the parameters fSWLimitPositive and fSWLimitNegative are set.

The template **LinDrive_V** will picture the drive in vertical shape.

RotDrive



For a rotary drive this picture will be displayed. The current position is shown by the position of the arrow and gets blue as soon as the drive is in regulation. It is a precondition for the use of the template, that parameter fPositionPeriod is set.

2.3 Drive Driver <BusInterfaceName>Drive.lib

- Drive-driver are responsible for the communication between IEC programs, especially the AXIS_REF-structures and the drives. They are CoDeSys libraries and contain at least the three functions mentioned in 2.2. Those libraries typically are provided by the manufacturer and must be included in the project.

- DummyDrive.lib is an example for drive-driver libraries and is provided with the SoftMotion-libraries. Even if this library does not serve real drives, it works according to the same principle.

2.3.1 SercosDrive.lib

With this library, which in turn uses the external library SercosBase.lib as an interface to the hardware, all Sercos-conforming drives can be controlled.

Similar to CAN there are function blocks for reading and writing of parameters:

- **SMC_ReadSercosParameter**
- **SMC_WriteSercosParameter**
- **SMC_ReadSercosList**
- **SMC_WriteSercosList**
- **SMC_ReadSercosString**

The precise range of functions is described in document SercosDrive.pdf.

2.3.2 SM_CAN.lib

- For each connected CAN-drive – contrary to Sercos - a separate driver is needed.
- However – if this is specified accordingly in the cfg-file – commonly for all CAN-drives in the PLC Configuration the baudrate and the number of the CAN-Controller (starting with 0) can be defined in the axisgroup dialog. In order to keep the bus deterministically, in a CAN-channel either I/Os or drives, but never both in time, is used. If the 3S-CANopen library should be used, that automatically will take the first CAN controllers and so for the axisgroup another one can be reserved.
- All CAN libraries created by 3S base on library SM_CAN.lib. It contains two modules which have practical meaning for the user, because they can be used to simply parameters of the drive: **SMC_ReadCANParameter** and **SMC_WriteCANParameter**. The functionality is similar to that of the modules MC_ReadParameter and MC_WriteParameter.

2.4 Variables of the AXIS_REF structure

During compiling the project CoDeSys will create a structure variable of type AXIS_REF (defined in SM_DriveBasic.lib, see Chapter 2.2). The structure AXIS_REF is used as an interface between application and drive interface. Via AXIS_REF cyclic and acyclic data are exchanged.

Most variables of the structure are not relevant for the user, but are used internally by the system. **The user always should use function blocks and never directly access the structure, at least not in a writing manner!**

No	Name	Data type	Initi value	comments
1000	nAxisState	INT	standstill	State of the Axis: 0: power_off 1: errorstop 2: stopping 3: standstill 4: discrete_motion 5: continuous_motion 6: synchronized_motion 7: homing

No	Name	Data type	Initi value	comments
1001	wControlType	WORD	PLC-Config*	Number indicates which parts of the structure are cyclically sended and received. 0: defined by param 1002-1008 1: SetTorque 2: SetVelocity,ActVelocity 3: SetVelocity,ActPosition 4: SetPosition,ActPosition 5: SetVelocity,SetPosition, ActVelocity,ActPosition 6: SetVelocity
1002 1003 1004	wCyclicDataS1 wCyclicDataS2 wCyclicDataS3	WORD	PLC-Config* or Init-FB of <BusInterfaceName> Drive.lib	Number of 3S parameters to be send each cycl
1006 1007 1008	wCyclicDataR1 wCyclicDataR2 wCyclicDataR3	WORD	PLC-Config* or Init-FB of <BusInterfaceName> Drive.lib	Number of 3S parameters to be received each cycle
1010	bRegulatorOn	BOOL	bRealDrive	regulator (power) on/off
1011	bDriveStart	BOOL	bRealDrive	set/unset halt
1012	bCommunication	BOOL	FALSE	TRUE: Drive answers
1015	bRegulatorRealState	BOOL	FALSE	State of the regulator
1016	bDriveStartRealState	BOOL	FALSE	State of the halt
1020	wAxisGroupId	WORD	PLC-Config*	Index of the axisgroup in the configuration
1021	wDriveId	WORD	PLC-Config*	Node number of the drive on the field bus
1022	iOwner	INT	0	Id-Number of the current owner (FB)
1023	iNoOwner	INT	0	Number of the previous and the current owners
1024	bMovedInThisCycle	BOOL	FALSE	has drive been moved in this IEC cycle ?
1025	fTaskCycle	REAL	PLC-Config*	Cycle time of task in ms
1026	bRealDrive	BOOL	PLC-Config*	TRUE: generated by Config; FALSE: generated by IEC
1030	bError	BOOL	FALSE	error occurred
1031	wErrorID	WORD	0	error id number
1032	bErrorAckn	BOOL	FALSE	Acknowledge error
1035	wFBErrorID	WORD	0	FB error id number
1051	dwRatioTechUnitsDenom	DWORD	PLC-Config*	Converstion of technical units in increments: Denominator
1051	dwRatioTechUnitsDenom	DWORD	1	conversion from technical units to increments: denominator
1052	iRatioTechUnits Num	INT	1	conversion from technical units to increments: numerator
1053	nDirection	MC_Direction	positive	-1 : negative (fSetVelocity < 0), 1: positive

No	Name	Data type	Initi value	comments
1054	fScalefactor	REAL	1	Conversion from bus unit to technical unit in techn. units per unit received on bus
1055	fFactorVel	REAL	1	Conversion from bus unit to techn. unit/s
1056	fFactorAcc	REAL	1	Conversion from bus unit to techn. units unit/s^2
1057	fFactorTor	REAL	1	Conversion from bus unit to Nm or N
1058	fFactorJerk	REAL	1	Conversion from bus unit to techn.units/s^3
1060	iMovementType	INT	1	0: Rotary (modulo) ; 1: Linear
1061	fPositionPeriod	REAL	1000	Length of Period for rotational systems in techn. units
1091	byControllerMode	BYTE	wControlType	1: Torque Control 2: Velocity Control 3: Position Control
1092	byRealControllerMode	BYTE	0	actual controller mode
1100/1	fSetPosition	REAL	0	Commanded position in technical units.
1101	fActPosition	REAL	0	Actual position in technical units
1105	fAimPosition	REAL	0	Position of destination (for some MC_FBs)
1106	fMarkPosition	REAL	0	internal position mark
1107	fSavePosition	REAL	0	internal position at begin of cycle
1110,11	fSetVelocity	REAL	0	Commanded velocity in technical units/sec
1111,10	fActVelocity	REAL	0	Actual Velocity of axes in techn. units./sec
1112,9	fMaxVelocity	REAL	100	Maximum velocity in techn. units/sec
1113	fSWMaxVelocity	REAL	100	Maximum velocity for implicit movements in techn. units/sec
1115	bConstantVelocity	BOOL	FALSE	Axes is driving with constant velocity
1116	fMarkVelocity	REAL	0	internal velocity mark
1117	fSaveVelocity	REAL	0	internal velocity at begin of cycle
1120	fSetAcceleration	REAL	0	Set acceleration in techn. units/sec^2
1121	fActAcceleration	REAL	0	Actual acceleration in techn.units/sec^2
1122,13	fMaxAcceleration	REAL	100	Maximum acceleration in techn.units/sec^2
1123	fSWMaxAcceleration	REAL	100	Maximum acceleration for implicit movements in techn.units/sec^2
1125	bAccelerating	BOOL	FALSE	axis is accelerating currently

No	Name	Data type	Initi value	comments
1126	fMarkAcceleration	REAL	0	internal acceleration mark
1127	fSaveAcceleration	REAL	0	internal acceleration at begin of cycle
1130	fSetDeceleration	REAL	0	Commanded deceleration in techn.units/sec^2
1131	fActDeceleration	REAL	0	Actual deceleration in technical techn.units/sec^2
1132,15	fMaxDeceleration	REAL	100	Maximum deceleration in techn.units/sec^2
1133	fSWMaxDeceleration	REAL	100	Maximum deceleration for implicit movements in techn.units/sec2
1135	bDecelerating	BOOL	FALSE	Axis is currently decelerating
1137	fSaveDeceleration	REAL	0	internal deceleration at begin of cycle
1140	fSetJerk	REAL	0	Commanded Jerk in technical units /sec3
1141	fActJerk	REAL	0	Actual Jerk in technical units /sec^3
1142,16	fMaxJerk	REAL	100	Maximum Jerk in techn. units /sec^3
1143	fSWMaxJerk	REAL	100	Maximum Jerk for implicit movements in techn. units/sec
1146	fMarkJerk	REAL	0	internal Jerk-Mark
1150	fSetCurrent	REAL	0	Set current (A)
1151	fActCurrent	REAL	0	Actual Current (A)
1152	fMaxCurrent	REAL	100	Maximum Current (A)
1153	fLimitCurrent	REAL	0	Maximum current for implicit movements in techn. units/sec
1160	fSetTorque	REAL	0	Commanded torque in Nm resp. N (linear)
1161	fActTorque	REAL	0	Current torque in Nm resp. N (linear)
1162	fMaxTorque	REAL	0	Maximum torque value in Nm resp. N (linear)
1200,2	fSWLimitPositive	REAL	0	Position limit in positive direction in techn.units/sec^2
1201,3	fSWLimitNegative	REAL	0	Position limit in negative direction in techn. units
1202	fCaptPosition	REAL	0	Capture position in techn. units
1205	bSWLimitEnable	BOOL	FALSE	Enable Software end switch
1204	bSWEndSwitchActive	BOOL	FALSE	Software end-switch active
1206	bHWLimitEnable	BOOL	FALSE	Enable / disable hardware end switch (to be used after overtravel)
1207	bCaptureOccurred	BOOL	FALSE	Capture signal occurred (acknowledged by writing)
1208	bStartCapturing	BOOL	FALSE	Start/stop capture of the current trigger

No	Name	Data type	Initi value	comments
1210	bStartReference	BOOL	FALSE	TRUE: start reference
1211	fReference	REAL	0	Reference position
1215	fOffsetPosition	REAL	0	Shift of zero point
1220	fFirstCapturePosition	REAL	0	Window start position for capture
1221	fLastCapturePosition	REAL	0	Window end position for capture
1222	tiTriggerInput	TRIGGER_REF		Description of the capture input
1223	bCaptureWindowActive	BOOL	FALSE	Capture limited to window
1230	dwPosOffsetForResiduals	DWORD	0	Internal variable for residual values' handling
1231	dwOneTurn	DWORD	0	Internal variable for residual values' handling
1232	fLastPosition	REAL	0	Internal variable for residual values' handling
1234	iRestNumerator	INT	0	Internal variable for residual values' handling
1235	iTurn	INT	0	Internal variable for residual values' handling
1236	dwBusModuloValue	DWORD	0	internal variable for residual values' handling
1237	dwPosOffsetForResidualsHoming	DWORD	0	internal variable for residual values' handling
1300	bDisableDriveInAxisGRoup	BOOL	FALSE	remove drive from axisgroup
1301	bErrorDuringStartup	BOOL	FALSE	will be set, as soon as an error occurs during startup
	pMS	POINTER TO BYTE	0	Pointer to hardware specific structure <BusInterface>_AXIS_REF

Each AXIS_REF structure variable behaves according to the PLCopen-Specification „Function blocks for motion control“, Version 1.0.

2.5 Parameterizing of the drive

Many important configuration data are stored in the drive. Though SoftMotion allows to set parameter values in the PLC Configuration, which are transferred during (see 2.1.3), this is difficult to do for the user, because normally he does not know, which parameters are to be transferred and which values they should get. Thus often for the start-up (also for several machines of the same series) and for an exchange of the drives drive-specific tools are needed. For this reason a functionality is integrated in SoftMotion which at least partly will take over this parameterization.

Thus there are the following alternatives:

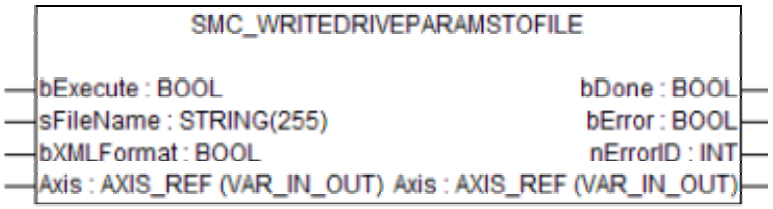
(A) The user configures the drives with the help of a tool provided by the manufacturer. He goes online with a SoftMotion project, where the drives are registered and calls a module reading all drive parameters and saving them to a XML-file on the PLC. This file will be re-load by the user and the data will be written to the appropriate configuration dialog in CoDeSys. Now all required

parameters are **saved in the project** and will be always transferred during start. Neither at the start-up of further machines nor at replacing a defect drive the start-up-tool would be needed.

(B) The user configures the drive with the help of the tool provided by the manufacture. In the application program he arranges that the user via a function block can save the drive parameters in an ASCII-file **on the controller**. In contrast to a. the parameters are stored in a file on the PLC and not with the application.

Both solutions have assets and drawbacks: If you want to change a parameter subsequently with (A), the project must be re-loaded. With (B) this is not necessary, however it must be taken care, that at each start-up of a machine either the parameter files get stored on the controller or the drive start-up-tool must be used.

SMC_WriteDriveParamsToFile



This module reads all configuration parameters of the drive and stores them to a file. As this is a file access, possibly blocking the processing of the application for several milliseconds, it may not be called in the motion task, but should be executed in a lower prioritized task.

Which parameters should be read the module learns from the drive-driver, which in turn this learns from the drive itself (Sercos) or which gets a default-parameter list (for CAN: see global variables list of the standard 3S drivers). For CAN drives an own list of the same format can be created and given to the drive via the following assignment:

```
<Drive>_MS.pParameterlist := ADR(<NewList>);
```

In-/Outputs of the module:

bExecute: BOOL

TRUE starts the module.

sFileName: STRING(255)

File name.

bXMLFormat: BOOL

If this variable is TRUE, the file will be created in XML format (and subsequently can be imported in the drive dialog), otherwise in text format (can be stored on the controller and get re-loaded and forwarded by the drive-driver during start-up).

Axis: AXIS_REF

Drive, whose parameters should be read.

bDone: BOOL

Action terminated.

bError: BOOL

Error occured.

nErrorID: INT

Error ID (see 9.2)

3 The CNC-Editor in CoDeSys

3.1 Overview

The CNC-Editor in CoDeSys allows to program multi-dimensional motions graphically and textually in parallel, following the CNC language DIN66025. For the CNC language see chapter 3.2 , concerning the text editor see chapter 0, programming examples you find in chapter 11.

Basically up to 9-dimensional motions can be realized, whereby only two dimensions are not interpolated linearly. Thus in two dimensions lines, circles, circular arcs, parables and splines can be programmed; the other directions merely get interpolated linearly.

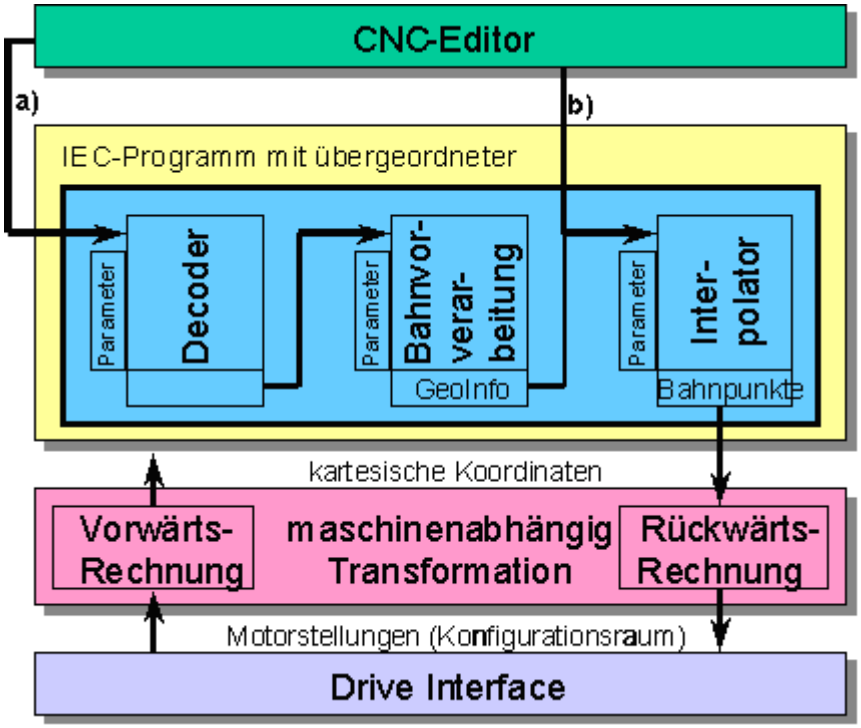
For each programmed path CoDeSys automatically creates a global data structure (**CNC Data**), which can be used by the IEC program.

This can be done in different ways:

a The CNC program is stored as an **array of G-Code-Words** and will be decoded during runtime of the PLC program with the aid of a decoder module. Thus for the particular path objects GEOINFO structure objects will be available. path-preprocessing modules (see SM_CNC.lib, e.g. Tool Radius Correction), afterwards interpolated, transformed and returned to the Drive Interface for the communication with the hardware. (see command 'Create program variable on compile')

b The CNC program is written as a **list (OUTQUEUE-Structure)** of GEOINFO structure objects to a data structure and thus can directly be fed to the interpolator. In comparison to a) by this method you can avoid calling the Decoder and the Path Preprocessing Modules. But in exchange you do not have the possibility to change the program during runtime. (see command 'Create OutQueue file on compile')

c The CNC program is written as described in a resp. b to the **file system** of the controller and is read and executed at runtime step by step. This method particularly is suitable for big programs, which cannot be kept completely in the memory.



3.2 Supported and extended elements of the CNC-language DIN66025

In order to provide an easy way for programming geometric motion profiles SoftMotion supports parts of the CNC language DIN66025. Since the whole SoftMotion concept is embedded in the much more powerful language IEC61131, **only those parts of DIN66025 are supported, which serve to create a path.**

Prescribed structure of a CNC program:

```
%
N<number> G<number> ...
...
N<number> G<number> ...
```

Example:

```
% example
N10 G01 X100 Y100 E100 F100 E-200
N20 G01 Z40 F20
N30 G03 X-100 R200 F100
...
```

A SoftMotion CNC program must start with a **"%"**. In the same line optionally can be added – separated by an empty space or by a TAB – the **program name**. The actual CNC program is composed of several **sentences**.

Each **sentence** (line) consists of as many **words** as needed.

A word consists of a letter (**word identifier**) and a subsequent number (e.g. G01; see also the list below). There is no case sensitivity and leading zeros are ignored (G01 = g1).

The first word of each sentence is built by the **sentence number** (N<number>), e.g. "N01". The sentence number currently does not have any effect, but is expected for conformity reasons. The words of a sentence are separated by empty spaces of TABs. They are processed from the right to the left. Thereby all words except for the **positioning command** (G<number>, e.g. "G02"; see the list below), will effect that the sentence number will be assigned to a variable as defined by the sentence letter. This variable finally will be accessed by the positioning command.

Each sentence only may contain one instruction, which must follow right to the sentence number. If you do not enter an positioning command in a sentence, automatically that of the last sentence will be added.

Each positioning command can be seen as a path object (line, arc, …). The velocity at which the path objects are interpolated, basically complies with the scheduled velocity (command speed), - acceleration and - deceleration. The Interpolator must make sure that these limits are not exceeded. The velocity during the transition of two adjacent objects is determined according to the following rules:

- One of both objects is a positioning (G0): Transition velocity = 0
- The angle between the tangents of the two objects at the transition is bigger than the angle tolerance: Transition velocity = 0
- Otherwise: The transition velocity is the lower command speed of the both path objects.

Basically a position command effects that there will be an interpolation from the target position of the last positioning command to the target position specified by the current positioning command. The first positioning command starts at the specified position (specified in the Decoder or CNC-Editor). If that position has not been defined, it will start at X=0, Y=0, Z=0. Additionally there is the possibility to set the position in the CNC program via G92. This is allowed at the beginning of the CNC program (there it will set the start position) as well as in the middle where it will result in a jump of the target position to the position defined by G92. If there are several successive G92-commands, the last will be regarded; the preceding ones will be skipped. If you however want to make sure that also the preceding G92 positions are given out (for the length of one cycle), you must insert command G1 with identic coordinates between. This means is used if the path between those positions is not of interest, but the target position should get there as fast as possible. The modules SMC_ControlAxisByPos in this case detect a jump of the target positions, stop the interpolator and interpolate each axis separately on the fastest way to the target position. Example:

```
G92 X100 Y100 (Set target position to 100/100)

G1 X100 Y100 (make sure a one-time output of the position)

G92 X50 Y100 (Set target position to 50/100)
```

A sentence starting wit the character "/" will be skipped during processing, if the option Step Suppress is activated .

Characters which are embraced by parenthesis "( )", will be interpreted as **comments** and do not have any effect on the programmed path. Nested comments are not supported.

The **line number** (N<number>) currently has no meaning, but is expected for conformity reasons.

All numbers except for that of the running order (G<number>) and the switch number (H<number>) can be floating values.

The **Switch functionality** enables the programmer to operate binary path-dependent switches. Basically always first the number of the switch must be specified ("H<number>"), then the switch position must be defined, either absolutely by using the word "L<position>" or relatively by using the word "O<position>". In the following example switch2 is turned off at position X=40/Y=25 (after a fourth of the object):

N90 G1 X20 Y20

N100 G1 X100 Y40 H-2 O0.25

Regard that a the number of possible switches within one path object is limited (MAX_SWITCHES).

Regard:	For each path object only a limited number of switch point switch actions (MAX_SWITCHES) can be processed. A switch point position only can be inserted in the CNC text editor! It will be displayed in the graphic editor as a green point on the path..

Additionally there is the possibility to use global variables instead of variables. These must be embraced by two $-signs (e.g. R$g_fVar$).

Word identifiers :

D	Tool radius (for correction G40-42 resp. for 'Round of Path' G50-51)
E	max. acceleration (>0) / deceleration (<0) [path-units/sec^2]
F	Velocity [path-units/sec]
G	Instruction (see below)
H	Switch on Switch point (>0) / Switch off (<0)
I	X-coordinate of the circle-/ellipse centre (G02/G03/G08/G09) – or X-coordinate of the parable-tangent-intersection-point
J	Y-coordinate of the circle-/ellipse centre (G02/G03/G08/G09) – or Y-coordinate of the parable-tangent-intersection-point
K	Direction of the ellipse main axis in mathematical sense (0° W, 90° S,...)
L	absolute switch position (see above, "H"), measured from start position (>0) resp. end position (<0) of the path object
O	relative switch (see above, "H") position [0..1]
P	Target position of the additional axis P
Q	Target position of the additional axis Q
R	Radius (G02/G03) – alternatively to "I","J" or length relation subaxis/main axis (G08/G09) [0..1]

S	Switch on (>0) / off (<0) the S-profile for linear axes 3: Z-axis, 7: P-axis, 8: Q-axis, 9: U-axis, 10: V-axis, 11: W-axis
U	Target position of the additional axis U
V	Target position of the additional axis V
W	Target position of the additional axisW
X	X-coordinate of the target position
Y	Y-coordinate of the target position
Z	Target position of the additional axis Z

Drive instructions:

G00	direct movement without tool contact, Positioning
G01	linear (straight) movement with tool contact
G02	Circle(-segment) clockwise
G03	Circle(-segment) counter clockwise
G05	Point of a cardinal spline
G06	Parable
G08	Ellipse(-segment) clockwise
G09	Ellipse(-segment) counter clockwise
G40	End of the tool radius correction
G41	Start the tool radius correction to the left of the work piece
G42	Start the tool radius correction to the right of the work piece
G50	End of round-off-path/slur-path function
G51	Start the slur-path function
G52	Start the round-off-path function
G60	End of the avoid-loop function
G61	Start of the avoid-loop function
G90	Start interpreting the following coordinate values (for X/Y/Z/P-W) as absolute values (default)
G91	Start interpreting the following coordinate values (for X/Y/Z/P-W) as relative values
G92	Setting the position without move
G98	Start interpreting the following coordinate values of I/J as absolute values
G99	Start interpreting the following coordinate values of I/J as values relative to the starting point (standard)

Please regard that the library "SM_CNC.lib " must be included to enable an error-free compilation of the project.

3.3 Start, Inserting and Managing of CNC Programs

The CNC-Editor is to be started in the ‚Resources' tab in the Object Organizer. A tripartite window will open, the title is '**CNC program list**'. In the left column there is a list of the existing programs. The

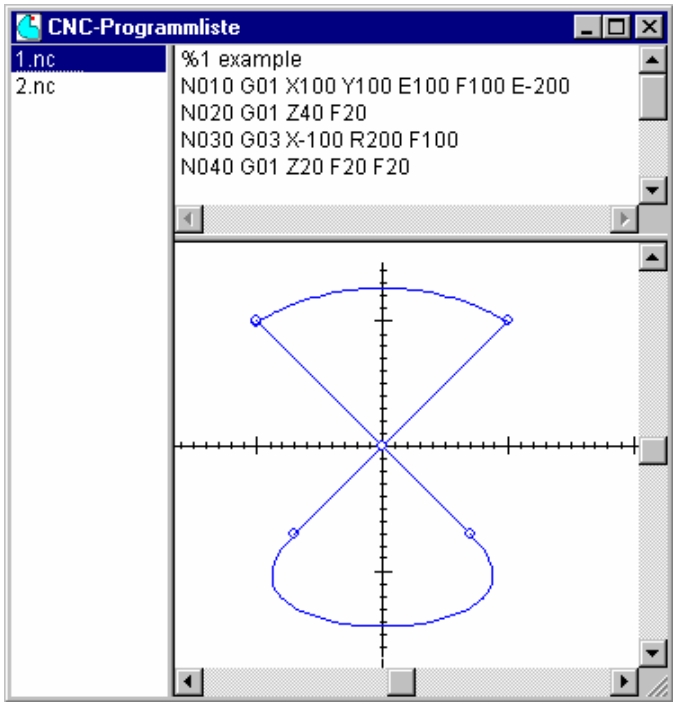
upper right window part serves as a text editor, where the CNC program can be written according to DIN66025. In the lower right part the program will be displayed graphically and finally you can modify the program it in the text or in the graphic part. In each case it will be updated automatically in the other part.

In the menu bar the item 'Insert' will be replaced by 'CNC program' as long as the CNC-Editor is active.

Insert a new CNC program:

Set the cursor to the CNC program list and choose command '**New CNC program**' in the menu 'CNC program' or in the context menu. A dialog will open where the name for the new program will be defined. By default the CNC programs get the name "_CNC<n>", where n is a running number, starting with "1". You can edit the default name in the dialog, but you cannot use an already existing name. After having closed the dialog with OK the new program name will appear (shaded) in the list. In the text editor the first program line is displayed: "% comment", the graphic editor still is empty.

The currently selected program can be edited in the Text editor as well as in the Graphic Editor.



Delete CNC program

Select the program in the CNC program list and choose the command '**Delete CNC program**' in the menu 'CNC program' or in the context menu. The program will be removed from the list and the focus will be set to the subsequent one.

Rename CNC Program

Select the program in the CNC program list and choose command '**Rename CNC program**' in the menu 'CNC program' or in the context menu. The dialog 'Program name' will open where you can edit the program name.

CNC-Program Info

Select a program in the CNC program list and choose command 'Info' in the menu 'CNC program' or in the context menu. The window **CNC program information** will open and provide information on the program.

Define queue size

This command opens the dialog **Size of queue data buffer**, where you can define the buffer size of the OutQueue. This is of main interest when only limited memory space is available for the NC function blocks in the IEC program, so that not all GeoInfo objects can be stored there and the ring buffer functionality must be used. Thus special effects can occur (e.g. slowing down at points without deviation), which you can simulate and reproduce by using this function.

Possible values: 5000 – 100 000 Bytes. With OK the settings are applied.

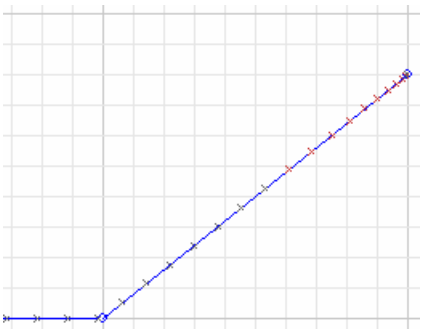
Define start position

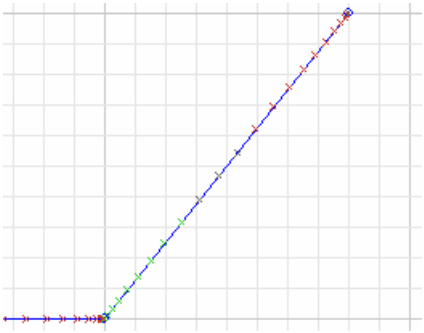
In the dialog **Define start position** you can set the coordinate values for the start position (default: 0) of the path for simulation purposes. For the following axes you can enter a start position: X, Y, Z, P, Q, U, V, W.

Set angle leeway for stop

In the dialog **Set angle leeway for stop** the sensitivity for a sharp bend of the path can be set. Insert the angle (0 .. 180) between the tangents of two path objects which will cause a stop.

Example: Maximum tolerance angle: 45°:





Move program

This command will open the dialog **translation vector** where you can define a vector by which the CNC program will be shifted. You can insert values for the following axes: X, Y, Z, P, Q, U, V, W.

Rotate program

Use this command if you want to rotate the current program. In the dialog **Rotation angle** insert the desired angle. The program will be rotated accordingly counter clockwise around the zero point.

Stretch program

This command opens the dialog **Stretch coefficient**. Insert here the factor by which the NC program should be stretched.

Invert direction

If you choose this command the path will be inverted in order to get processed in the reverse direction. The switch positions will stay as they were before.

Split object

This command opens a dialog where you can define for the currently selected path object a **Split position**. Enter any value between 0 (start position of the current object) and 1 (end position) to mark the desired position where the object should be split to two objects.

Example: Object N10 will be divided at position 0.5:

```
...
N0 G01 X123.000000 Y73.550000
N10 G01 X40.0 Y50.0
...
```

Result: new object (additional position) at X=20

```
...
N0 G01 X20.000000 Y40.000000

N10 G01 X40.000000 Y50.000000

N20 G01 X123.000000 Y73.550000
...
```

The resulting new Y-subposition will be automatically adapted according to the path progression.

Read CNC program from file

You can load a CNC program which is stored in a file in ASCII format. The standard dialog for opening a file will be available, where you can select the desired *.txt-file. In the next dialog you must insert a name for the program before it can get loaded to the editor.

Write CNC program to file

You can write the current CNC program to a file in ASCII format (*.txt). If the file you define is existing already, CoDeSys will ask for a confirmation.

Import DXF file

Use this command if you want to import a DXF file to your CNC program. The standard dialog for opening a file will be opened where you can browse for the desired DXF file. Then the dialog DXF import options will open, where you enter a base name for the CNC program(s) (**Program base name:)** which should be created from the DXF file and where you activate one of the following options:

- **One NC program for whole DXF file**: all paths described in the DXF file will be written to one CNC program
- **One NC program for each DXF layer**: for each DXF layer a separate CNC program will be created
- **One NC program for each coherent segment: for each coherent path segment**: a separate CNC program will be created. Due to the fact that in a DXF file the single path objects are stored without a certain order, CoDeSys tries to connect the objects in a way that results in a coherent path.

Write outqueue to file

. With this function you can convert the complete CNC program to an OutQueue, which is a list of GEOINFO structures objects, and save this list in a file which then can be downloaded to the controller's file system and can be read there during run time (see chapter 3.1, option (b). It is recommended to proceed in this way, if you have CNC programs, which are to big for the global data memory of the controller or which must be exchanged without a change of the CoDeSys project.

3.4 CNC Text editor

The text editor is in the upper right part of the 'CNC program list' window. Here you can enter and modify a CNC program according to DIN66025 (see Chapter 3.2 for the supported elements). The program will be displayed accordingly in the graphic Editor and can be modified there. In this case the text part will be updated vice versa.

For working in the editor there are commands available in the menus 'CNC program' and 'Extras'. See a description in chapter 3.5, Graphic Editor.

(Hint: Have a look to the programming examples in Chapter 1)

By pressing <F2> the input assistant will open and you can select global variables to be inserted in the CNC program. The variables will be displayed in the graphic editor by their initial value (if available).

Note:	Please regard, that references to global variables in the Decoder module will be evaluated as soon as a rising edge in input 'Execute' occurs.

If you have chosen variant b, which makes CoDeSys creating a ready OutQueue-structure during compilation, then the reference on the global variable of course will be replaced by the initial value already during compilation which makes the use of global variables senseless in this context.

3.5 CNC Graphic Editor

The graphic editor is in the lower right part of the CNC program list window. On the one hand it visualizes the CNC program which is described in the text editor, on the other hand also in the graphic editor you can modify the program using the mouse and these changes will be updated automatically and immediately in the program text in the upper window part.

Display:

A **coordinate system** is displayed. There are pitch lines marking the intervals on the coordinate axes and additionally a light-grey colored grid can be displayed, which can switched on and off via the command **Show grid** (Extras menu resp. context menu).

By keeping pressed the left mouse-button, the display of the CNC program can be moved as desired. Using mouse-wheel and <Ctrl>-key the zoom factor can be changed.

Positionings (G00) are displayed green-colored, all other **elements** blue-colored. The **currently marked object** (the cursor is placed on the corresponding code line in the text editor), is displayed red-colored.

For **Splines** (G05) the convex mantle of the cubic poynoms is displayed light grey-colored.

3.6 Commands and Options in the CNC-Editor

The **'Extras'** menu provides commands and options for the working and the display in the editor. (Of course you also can program by entering the corresponding instructions in the text editor). Additionally in the toolbar the appropriate buttons are available. An activated option gets marked by a check in the 'Extras' menu and the button in the toolbar appears pressed.

You can choose one of the following edit modes:

Select mode

Line-Insert Mode

Circle CW Insert Mode

Circle CCW Insert Mode

Spline Insert Mode

You can set the view and path correction options via the following commands:

Fit to Screen

Renumber program

Show grid

Convert splines/ellipses to lines

Step Suppress

Show Interpolation points

Tool radius correction

Round off path

Slur path

Avoid loop

Set epsilon values

Note:	It is possible to choose more than one of the options 'Tool correction', 'Round off path', 'Slur path' and 'Avoid loop'. Thus the effect of series connected path-preprocessing elements can be simulated. Only the preprocessing options 'Tool correction' and 'Round off path' resp. 'Slur path' cannot be activated at the same time.

'Extras' 'Select Mode'

If this option is activated, you can select a graphic element in the CNC-Editor by a mouse-click. A selected element will be marked red-colored and in the text editor the corresponding line will also be marked. By a click on the end point of an element and keeping the mouse-button pressed the element can be shifted by moving the mouse.

'Extras' 'Line Insert Mode'

If this option is activated, a mouse-click within the editor panel will insert a G01 line element. The line will start behind the currently selected (red marked) element. The mouse position defines the second (end) point of the line.

'Extras' 'Circle CW Insert Mode'

If this option is activated, a mouse-click within the editor panel will insert a G02 circle element, used for a motion of clockwise rotation. The new element will be inserted behind the currently marked (red-colored) element. The mouse position defines the second (end) point. The radius of the circle will be set to 100 by default and – if necessary – must be modified in the text editor.

'Extras' 'Circle CCW Insert Mode'

If this option is activated, a mouse-click within the editor panel will insert a G03 circle element which is used for a motion of counter clockwise rotation. The new element will be inserted behind the currently marked (red-colored) element. The mouse position defines the second (end) point. The radius of the circle will be set to 100 by default and – if necessary – must be modified in the text editor.

'Extras' 'Extras: Spline Insert Mode'

If this option is activated, a mouse-click within the editor panel will insert a Spline. That will be placed behind the currently marked (red-colored) element. The mouse position defines the second (end) point of the spline.

'Extras' 'Extras: Fit to Screen'

If this option is activated, the visible part of the window will show the complete NC program.

'Extras' 'Renumber program'

This command automatically renumbers the program, assigning new line numbers (N<number>) in decimal steps.

'Extras' 'Show grid'

If this option is activated in menu 'Extras', a visible grid will be added to the graphic editor.

'Extras' 'Convert splines/ellipses to lines'

Splines and ellipses need a lot of computing time for interpolation. In order to reduce this time, use this command to approach all splines and ellipses of the NC program by a number of lines. Thus for designing the path splines and ellipses can be used but those must not be computed during run time.

The command opens a dialog providing two options for the conversion:

a) **length dependent**, i.e. per x length units (number x can be inserted in the dialog) of the spline/ellipse a line will be created, or

b) **angle dependent**, i.e. the original object will be partitioned in a way that the arising lines will include angles lower then x (angle x [degrees] can be inserted in the dialog).

Hint: Using the default settings will effect, that at the interpolation of the line – contrary to spline/ellipse – after each part of line there will be a deceleration to velocity 0. You can avoid this by increasing the angle tolerance correspondingly.

'Extras' 'Extras: Tool radius correction'

If this option is activated and if in the CNC program the start (G41/G42) and the end (G40) of the of the path segment, where the correction should be done, are defined as well as a tool radius (D<angle>) , then the accordingly corrected path will be displayed. This menu item corresponds to the SMC_ToolCorr module which is part of the library SM_CNC.lib. The original path will be colored light-grey. Positionings of the corrected path are "blind" positions and therefore are colored dark-yellow.

'Extras' 'Slur path'

If this option is activated, the programmed path will be displayed, showing the effect, the function block SMC_SmoothPath (SM_CNC.lib) has on the originally programmed path. That creates a slured path through the cubic polynom (spline). Preconditions: In the CNC program the start (G51), the end

(G50) of the path segment, where the correction should be done, as well as the rounding radius (D<angle>) must be set. The original path will be displayed as a reference, colored light-grey.

'Extras' 'Round off path'

If this option is activated, the rounded-off path will be displayed, showing the effect, the function block SMC_RoundPath (SM_CNC.lib) has on the originally programmed path. Preconditions: In the CNC program the start (G52) and the end (G50) of the path segment, where the correction should be done, as well as the rounding radius (D<angle>) must be defined. The original path will be displayed as a reference, colored light-grey.

'Extras' 'Avoid loop'

If this option is activated, that path will be displayed which results if loops are cut. This means that, if the path crosses itself, at the crossing points the loop part will be deleted, which shortens the path. So loops can be avoided. This command corresponds to the effect of the funktion block SMC_AvoidLoop (SM_CNC.lib). Preconditions: In the CNC program the start (G61) and the end (G60) of the path segment, where the correction should be done, must be defined. The original path will be displayed as a reference, colored light-grey.

'Extras' 'Extras: Step Suppress'

If this option is activated, all lines of the text editor starting wit "/" will be ignored.

'Extras' 'Extras: Show Interpolation points'

If this option is activated, interpolation points will be displayed in 100 ms pulse, this means that the tool positions will be indicated by small grey crosses every 100 ms. Thus a rough estimation of the velocity behaviour (fast = long distances, slow = small distances) is possible.

'Extras' 'Set epsilon values…'

The internal check of a value x for zero must be replaced by an examination for $x < \varepsilon$ because of the inaccuracy of a floating point calculation. The size of the ε-value eventually (e.g. if 32 bit floating point values are used instead of 64 bit, or at import of a CNC program with limited accuracy) must be adapted. For this purpose use the dialog **Zero tolerance values**, which will be opened by the here described command.

Please regard: For standard use any modification of the tolerance values should not be necessary and should be avoided.!

3.7 Automatic structure filling in the CNC-Editor

As soon as the IEC program is compiled, automatically a global variables folder "CNC Data" will be created.

There the CNC programs will be stored in homonymous data structures.

According to chapter 3.1, 'Text editor - Overview' you can choose between three possible options a-c for each CNC program, which can be selected in the menu 'CNC-Program' in the text editor:

(1) Create program variable on compile

This option corresponds to variant a) (see chapter 3.1)

The CNC program is stored to a structure SMC_CNC_REF which is defined in the library SM_CNC.lib. This structure

Obwohl diese Variante erhöhten online-Rechenbedarf hat, bietet sie die Möglichkeit, Variablen im NC-Programm zu verwenden, bzw. Modulationen des NC-Programms durch das SPS-Programm vorzunehmen.

(2) Create OutQueue file on compile

This option corresponds to variant b) (see chapter 3.1)

The CNC program is stored in a structure SMC_OUTQUEUE which is defined in the library SM_CNC.lib. This structure variable can be passed on directly to the Interpolator module.

So no variables can be used in the path, but the advantage of this method is the minimized demand of online ressources.

(3) Don't compile

This option corresponds with option (b described in chapter 3.1. You have stored the program as an ASCII or as an OutQueue file in the file system of the controller and you want to read it at run time by one of the modules described in 10.2, 'CNC function blocks'. For this reason it should not be added to the IEC data.

4 The CAM-Editor

4.1 Overview

The SoftMotion CAM disc (CAM-Editor) is integrated in the CoDeSys programming interface. Here you can graphically and tabularly create programs of electronic CAM discs and CAM switches, for which CoDeSys automatically will create global data structures (CAM Data) during the compilation of the project. This structures can be accessed by the IEC program.

For the preprocessing of the CAMs in the IEC program the functions and function blocks defined by PLCopen are used. (Library SM_PLCOpen.lib).

(Hint: Have a look at the programming examples in Chapter 11)

4.2 Definition of a CAM for SoftMotion

A CAM describes – seen in a simplified manner - the functional dependence of a value (Slave) on another (Master).

In order to describe this dependency the **master-axis** is divided up in different segments. For each segment (interval [a,b]) CoDeSys provides two possibilities to image a functional mapping of the master axis on the **slave-axis**:

- **Line**: The dependency is described by a linear image. In this kind of segment the first derivative (velocity) is constant, according to the slope of the line, the second derivative is 0.
- **5 exponent Polynom**: In this kind of segment the dependency is described by a 5-exponent polynomial. Thus the first and the second derivative become 4- and 3-exponent polynomials.

The functions in these segments must follow on each other in a way that at the transition points as well the function value as also at least the first and second derivatives are continuous.

In the CAM-Editor single base points and lines can be inserted. The remaining sections between will be filled up automatically by the editor with 5-exponent polynomials. Thereby the requirements concerning continuity and differentiation are regarded.

Along a line the function value, the first derivative (velocity, in this case constant) and the second derivative (acceleration, in this case always 0) are defined. A point however can be defined with any first and second derivative.

Additionally the user has the possibility to place tappets, i.e. binary position switches, on the CAM disk.

4.3 Starting the CAM-Editor and Inserting a new CAM

Start

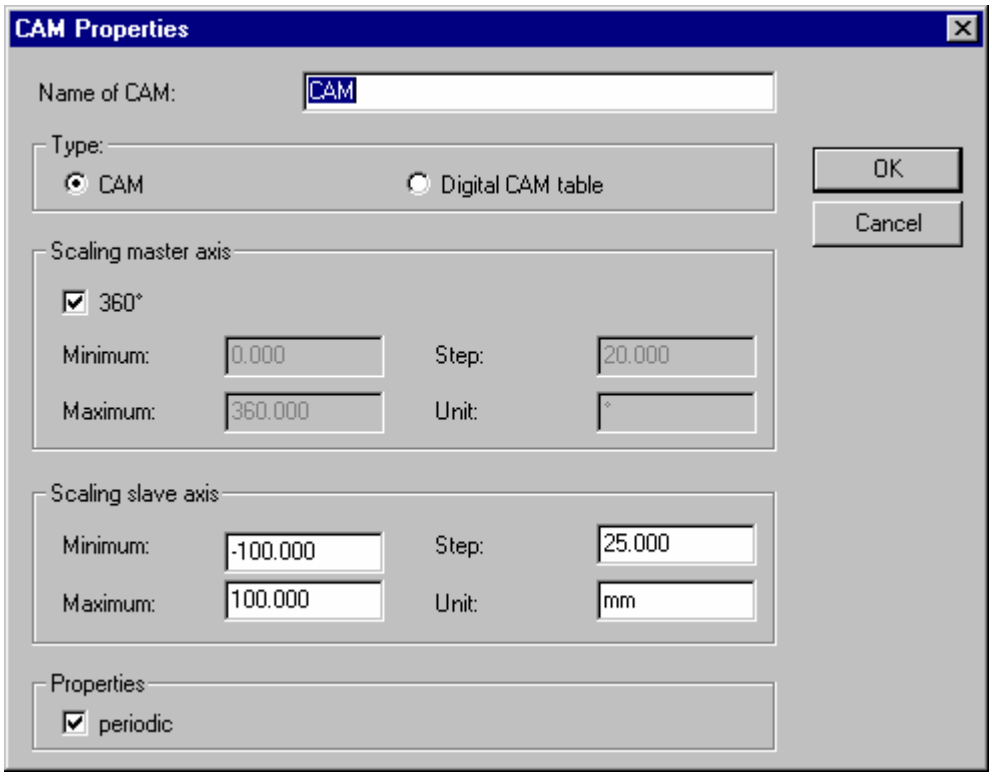
The CAM-Editor is started in the 'Resources' tab. A tripartite window, titled 'CAM program list' will open. As long as no CAM has been defined, the window is empty.

In the lower right part of the window three different types of display can be chosen (see Menu Extras): Visualization of the first (blue curve, velocity) and second (green curve, acceleration) derivatives, Visualization of a table showing all CAM elements (points/lines) or Visualization of a table showing all CAM switches (tappets). The tables can be edited.

The menus 'Insert' and 'Extras' (see chapter 4.4.1, General editor settings) provide commands for the creation and editing of CAMs.

'Extras' 'Create new CAM'

Choose the command '**New CAM**' in the 'Insert' menu or in the context menu and do the desired settings:



Name of CAM: A name for the new CAM

Type: A CAM contains CAM elements (points, lines) as well as tappets. A Digital CAM table only contains tappets.

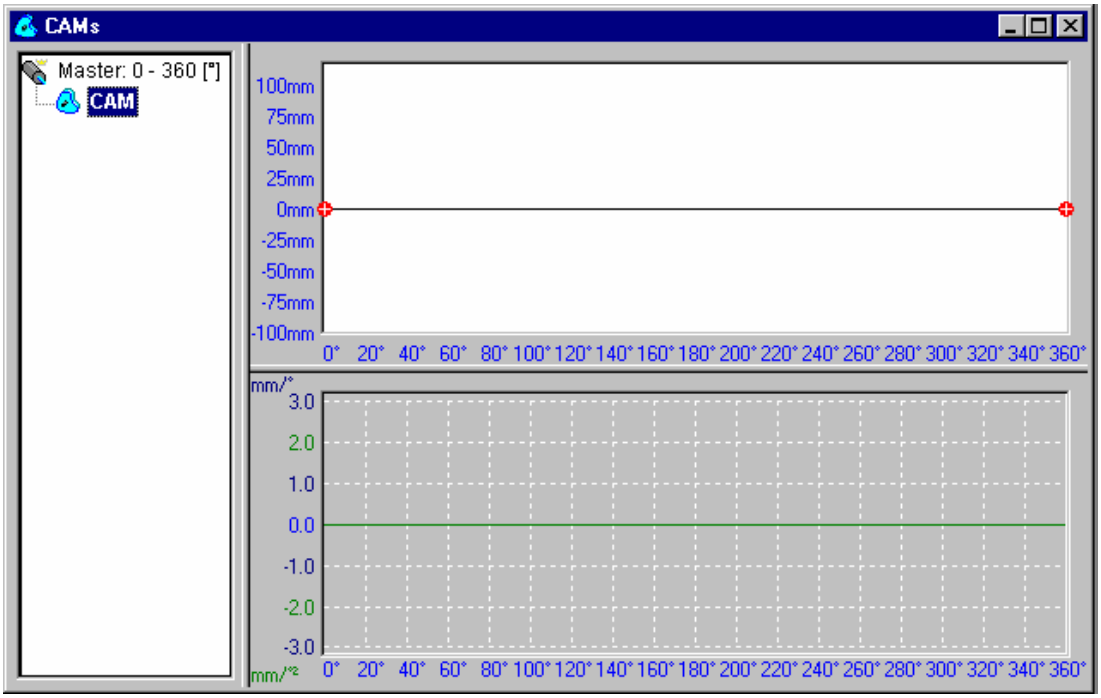
Scaling master axis: Define here the scaling of the master axis. If option **360°** is activated, the settings **Minimum**, **Maximum**, **Step** and **Unit** will be set automatically (0, 360, 20, °); otherwise you can define them manually.

Scaling slave axis: Define here the scaling of the slave axis. See for the defaults in the picture shown above.

Properties: If the option **periodic** is activated, it will be guaranteed that the function values and the first and second derivatives of the start and end point of the CAM are matching. Any modifications of the endpoint parameters, which have been done during the editing of the curve will be ignored.

Close the dialog with **OK** to confirm the settings.

Hereupon in the CAM list on the left side of the window the name of the new CAM appears. As long as this entry is marked, the CAM is displayed in the Editor and can be edited. In the right part of the window the new CAM will be visualized. You see the horizontal blue master axes, the vertical blue position axis (slave) in the upper window and the velocity- (dark-blue) and acceleration scale (green) in the lower window. The following picture corresponds to the default settings in the 'CAM Properties' dialog:



For editing the above described settings the properties-dialog for the currently marked CAM can get reopened anytime by a double-click on the entry in the CAM list or by using the command 'Settings', which is available in the 'Extras' menu or the context menu.

CAM tree

In the left part of the editor window a tree is displayed, showing all CAMs and CAM switches (tappets) . These elements always are sorted in a way, that all elements which have the same master scaling, that means which potentially refer to the same axis, have the same "father".

4.4 Editing a CAM

In the left column of the editor select the CAM which you want to edit. For this purpose perform a mouse-click on the entry, which hence will be displayed selected (shaded) and displayed in the editor windows.

By simultaneously pressing the <Ctrl>-key and performing a mouse-click on one or several further CAMs which have the same master (see chapter 4.3, CAM tree), those will be displayed additionally.

4.4.1 General Editor Settings

(For a description of the corresponding commands in the 'Extras' and 'Insert' menus see Chapter 4.4.3.)

The Edit mode can be selected in the **Insert** menu or by the corresponding button in the tool bar :

Select elements

Insert point

Insert line

Insert tappet

Editing the general properties of the CAM disc: In order to modify the settings which you have made before (when creating the CAM) in the dialog '**CAM Properties**', (see Chapter 4.3) use the command **Settings** in the **Extras** menu.

Display of the CAM: For this the command "**Show complete CAM**" in the **Extras** menu.

Display mode of the lower part of the editor window: Choose one of the following types by the corresponding command from the 'Extras' menu or by the button in the tool bar:

Show velocity/acceleration: In the lower window the first (blue) and second (green) derivation of the CAM will be visualized.

CAM as table: The lower window shows the CAM elements (points/lines) and their properties in an editable table.

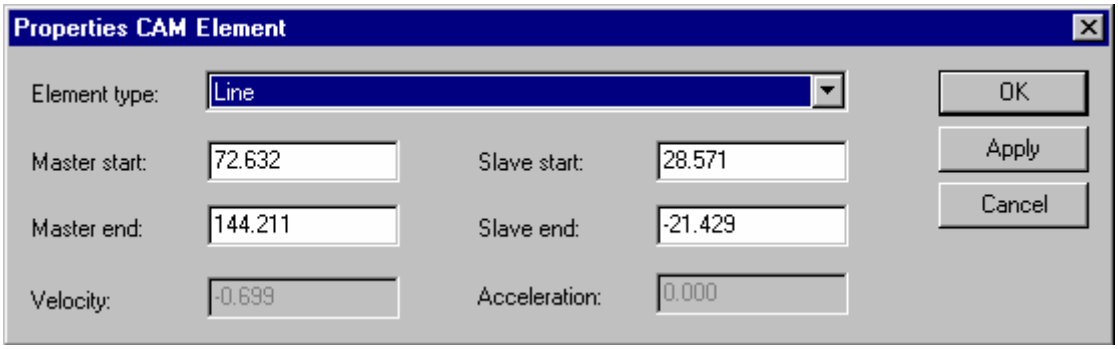
Tappets as table: The lower window shows the CAM switches (tappets) and their properties in an editable table.

4.4.2 Editing the properties of a particular CAM element

The attributes of a single path object can be modified in the 'CAM Element Properties' dialog or by selecting and moving the element in the editor window:

1. *In the 'CAM Element Properties' dialog:*

   Point, Line: By a double-click on the element in the CAM editor window the dialog '**CAM Element Properties**' can be opened to edit the following properties by numeric inputs:



   Element type: "Line" resp. "Point"; if a line is "switched" to be a point, then it will automatically get a certain length; if a point is "switched" to be a line, automatically the coordinates of the start point of the line will be applied.

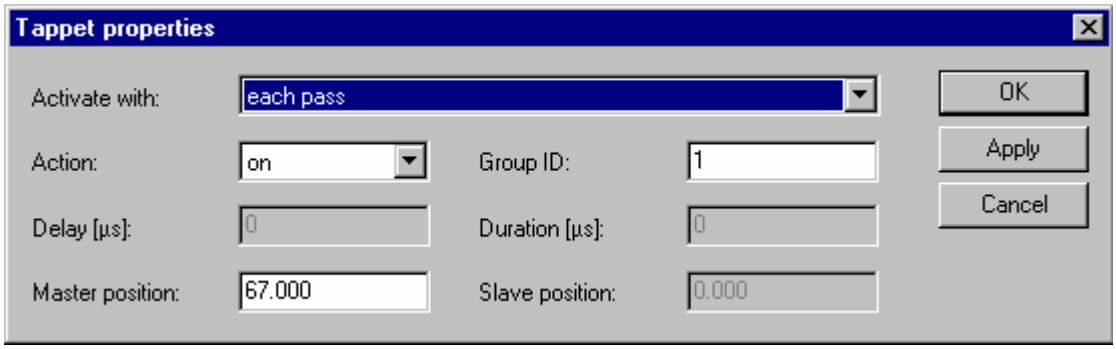
   Master start, **Master end**: Start- and end values on the X-axis (Master) (unit see 'Extras' 'Settings')

   Slave start, **slave end**: Start- and end values on the Y-axis (Slave) (unit see 'Extras' 'Settings')

   Velocity: (only for Points)

   Acceleration: (only for Points)

Tappet: By a double-click on the element in the CAM editor window the dialog **Tappet properties** gets opened, where the following settings can be made:



Activate with: The tappet gets activated, this means that the boolean variable (tappet bit), which is assigned to the tappet-GroupID (see below), will be set to TRUE when the CAM is run through; one of the following options can be set to define when exactly this should happen:

positive pass: if the CAM is passed through from the left to the right; after confirming with 'Apply' the green arrow above the tappet symbol will point to the right

negative pass: if the CAM is passed through from the right to the left; after confirming with 'Apply' the green arrow above the tappet symbol will point to the left

each pass: at each pass of the CAM; after confirming with 'Apply' the green arrow above the tappet symbol will point to the right and to the left

Action: One of the following options can be set to define, which effect the activation of the tappet should have on the action which is assigned in the project:

on: The action will be started (the tappet-bit will be set tot TRUE); The tappet symbol will be filled green

off: The action will be stopped (the tappet-bit will be set tot TRUE); The tappet symbol will be filled red

invert: If the action is currently active, it will be stopped; if it is currently inactive, it will be started; (the tappet-bit gets inverted); The tappet symbol will be filled yellow

timed in: The action will be started with the values, which are given in the fields 'Delay' and 'Duration'; The tappet symbol will be filled cyan-colored

Group ID: Identification number (INT) of the tappet which serves to reference the tappet in the project; several tappets can get the same GroupID and thus get "grouped" for the purpose that the assigned action would serve the same digital switch.

Delay [µs]: Period of time, which should be waited before the action assigned to a tappet gets started after the tapped was passed (after which the tappet-bit should be set tot TRUE). (only if action = timed in)

Duration [µs]: Define here, how long the action, which is assigned to the tappet, should stay active (how long the tappet-bit should stay TRUE). (only if action = timed in)

Master position: X-position of the tappet

Slave position: Y-position of the tappet, not editable, because determined by the curve progression

The settings in the properties dialogs can be confirmed by **OK** or **Apply**. The CAM curve in the editor will be displayed correspondingly. OK will also close the dialog, whereas it will stay open with Apply.

(Editing the properties of particular CAM elements)

2. *By selecting and moving in the editor window*

   An element can be selected by a mouse-click. To move an element, keep the mouse-key pressed and move the cursor to the desired position. This will cause an modification of the corresponding values in the properties dialog of the element:

Point: If you select a point in the CAM editor, a little red square will be displayed representing the slope (Velocity). By moving this square (point on the square with the cursor, so that the cursor symbol will be displayed as a cross, and then move the cursor) the velocity value for the point can be modified. The slope will be displayed with the aid of an auxiliary tangent. Also the point itself can be moved.

Line: If you select a line in the CAM editor, little red squares will be displayed at the end points. You can change the slope (velocity) of the line by moving one of the end points (point on the square with the cursor and move the cursor); you can move the line without changing the slope by pointing on the line between the end points and moving the cursor.

Tappet: If you select a tappet in the CAM editor the frame of the tappet symbol (square) gets red. The tappet can be moved along the CAM curve by moving the cursor.

4.4.3 Commands of the 'Extras' and 'Insert' Menus

'Extras' 'Settings'

This command opens the dialog 'CAM Element Properties', which you have edited during creating the CAM. (See the picture in Chapter 4.3). Here you can modify the scaling and the units.

'Extras' 'Show complete CAM'

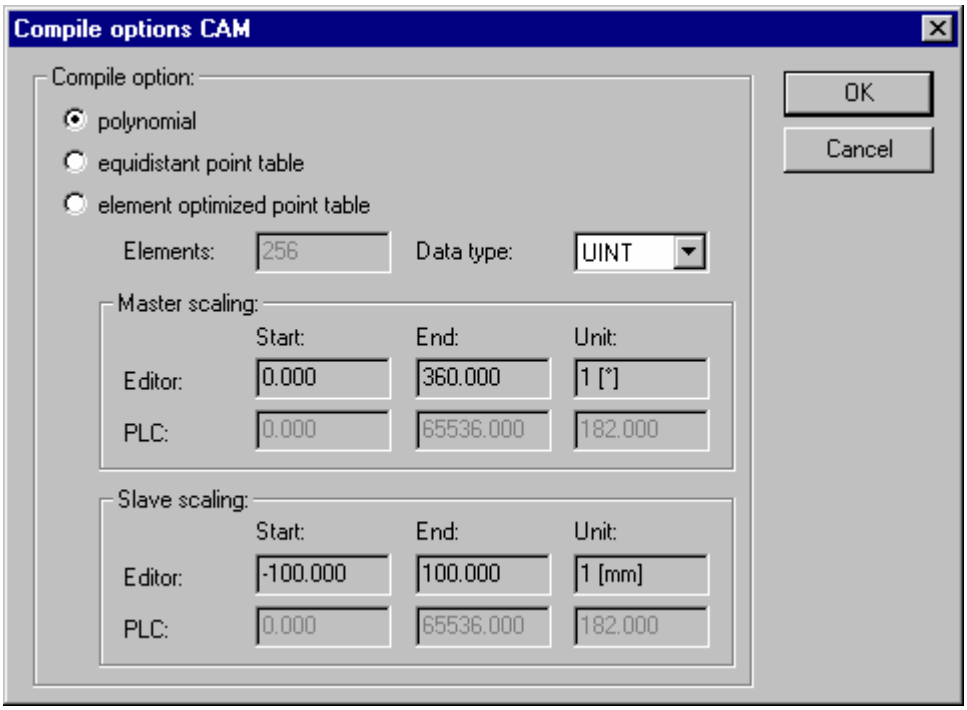
If this option is activated (a check is displayed before the command in the Insert menu, the button in the tool bar appears "pressed"), additionally the '5-exponent polynomials', which are filling the intervals between the CAM elements points, lines and/or tappets, will be displayed. Otherwise just the elements are visible.

Show bounds

If this option is activated (a check is displayed before the command in the Insert menu, the button in the tool bar appears "pressed"), besides the CAM and their derivatives also its bound values are displayed (Maximum/Minimum).

Compile options

This command opens a dialog where the compilation of the CAM can be configured.



Basically there are three modes of compilation:

1. **polynomial**: During compilation structure variables of type MC_CAM_REF are created. They contain for each segment the description of the 5 exponent polynom which describes the CAM. Structures of this type are used as input of the MC_CamIn module. The structure is part of the library SM_DriveBasic.lib.

2. **equidistant point table**: According to the settings in the lower part of the dialog a table of base points is created. The table is of type SMC_CAMTable_<datatype>_<number of elements>_1. The position-array of the dialog contains the slave-values of the CAM referring to the master-values, which are arranged evenly on the defined range of the master axis. The first value of the table refers to the slave position at the master minimum of the CAM. The last value at non-periodic CAMS refers to the slave position at the master maximum. At periodic CAMs this value needs not to be re-written, because it is the same as that at the master minimum; for this reason the intervals are made slightly narrower and the last value of the table describes the slave position at Master. End - (Master.End-Master.Start)/Number of elements.

3. **element optimized point table**:According to the setting in the lower part of the dialog a two-dimensional (typically not equidistant) base point table of type SMC_CAMTable_<Datatype>_<Element number>_2 is created. The table which is contained, describes pairs of master- and attached slave-positions. The partition is done in a way, that elements with constant velocity (lines) each only get one base point at start and end. The remaining base points are arranged as evenly as possible on the residual CAM.

4. **don't compile**: No global variables are created for the CAM. This option mainly is used if the CAM should be loaded from the file system at run time (see 10.3, CAM function blocks), e.g. because it must be changed without making necessary a change of the running CoDeSys project.

The Master- and Slave **scaling** only is of interest for the base point tables. You can define the scaling of the master and the slave axis either via start and end value, or via start value and unit.

'Extras' 'Write CAM to file'

This command opens a file selection dialog, where a *.CAM file can be defined, to which the currently edited CAM can be written. This file can be read at run time by the function block SMC_ReadCAM (see 10.3, CAM function blocks) and can be converted to a standard data structure Depending on the currently set compile option the CAM will be stored in polynomial, equidistant or element optimized format.

'Extras' 'Read CAM from file'

This command can be used to import a CAM description to the CoDeSys CAM editor by reading it from a *.CAM file. After the desired file has been selected, the dialog 'CAM properties' will be opened, where a name for the CAM must be defined and the scaling of the slave axis must be adapted. Due to the fact that during the creation of a CAM only that information is output which is needed for the execution of the CAM, a read CAM can be different from the original.

'Extras' 'Export CAM as ASCII-Table'

This command can be used to export the current CAM in an ASCII text file. The number of points can be specified. The start and end point always will be contained as first resp. last point. A text file with the following structure will be created:

<Master-Position>;<Slave-Position><CR><LF>

The resulting text file for example can be imported in other programs and can be used for the layout of drive line.

'Insert' 'Select elements'

Use this command to switch on and off the select mode. As long as the select mode is activated (a check is displayed before the command in the Insert menu, the button in the tool bar appears

"pressed"), you can select an CAM element by positioning the cursor on the desired element (point, line, tappet) and pressing the left mouse-button. Hereupon the corresponding position marks (little red squares) will be displayed and the element can be edited.

'Insert' 'Insert point'

Select this command in the 'Insert' menu or press the button in the tool bar to insert a new point in the CAM. A point-symbol will be added to the cursor. Position the cursor where you want to set the new point and press the left mouse-button. The point symbol (red filled circle with a hair cross) will be displayed on the curve and additionally a horizontal tangent, representing the slope, will be shown. If you loose the mouse-button, the point will be inserted and automatically it will be switched to the select mode ("Select elements ".

If you keep the left mouse-button pressed during inserting the new point you can immediately modify the slope of the tangent (Velocity) by moving the mouse.

'Insert' 'Insert line'

Select this command in the 'Insert' menu or press the button in the tool bar to insert a line in the CAM. A line-symbol will be added to the cursor. Position the cursor where you want to start the line (left end point) and keep the mouse-button pressed. Move the cursor to the desired endpoint which must be to the right of the start point and to the left of the next defined point, line or tappet. As soon as you loose the left mouse-button the end point will be applied and automatically it will be switched to the select mode ("Select elements ".

'Insert' 'Insert tappet'

Select this command in 'Insert' menu or press the button in the tool bar to insert a tappet in the CAM. A tappet-symbol will be added to the cursor. Position the cursor on the desired position for the new tappet. You do not have to care for the Y-Position, for the tappet will be displayed appropriately to the chosen X-value on the CAM curve. As long as you keep the left mouse-button pressed, you can move the tappet along the curve. As soon as you loose the button, the tappet will be inserted and automatically it will be switched to the select mode.

4.5 CAM data structures

During compilation of the project a global variables list, named **CAM Data**, will be created from the CAM data produced in the CAM-Editor. The description of each single CAM will be written to a structure of type MC_CAM_REF corresponding to the settings in dialog 'Compile options' and by that made accessible for the IEC-program resp. the CAM preprocessing functions and function blocks. In order to get an error-free compilation the appropriate structure definitions must be available in the IEC program.

(Hint: Have a look at the programming examples in chapter 11)

Of course the corresponding data structures also can be created resp. filled from the IEC program during run time. For this reason they will be described more detailed in the following. Not mentioned variables of the structure are used only internally.

MC_CAM_REF

This data structure represents a generic CAM and contains the following elements:

wCAMStructID: WORD

This variable, which always has a fix value, is used to check whether the data structure, given to the module as an input, is a MC_CAM_REF structure.

xStart, xEnd: LREAL

Domain of the CAM. Start- and end position of the master.

byType:BYTE

This variable describes the CAM type, i.e. the way in which the CAM is represented.

1: equidistant, 1-dimensional table of slave positions

2: non-equidistant, 2-dimensional table of master/slave-point pairs

3: polynomial description on particular points consisting of masterposition, slave-position, -velocitiy and -acceleration (XYVA).

byVarType:BYTE (nur für byType=1 oder byType=2)

Variable type, the curve table consists of:

0: INT

1: UINT

2: DINT

3: UDINT

4: REAL

5: LREAL

nElements:INT

Number of elements, so depending on type number of slave positions, master/slave positions or XYVA points.

byInterpolationQuality:BYTE (nur für byType=1 oder byType=2)

Fine interpolation degree: 1: linear (default), 3: cubic

pce: POINTER TO BYTE

Pointer on the actual data element; depending on type:

Type	
1 (equidistant)	SMC_CAMTable_<VarType>_<nElements>_1
2 (non-equidistant)	SMC_CAMTable_<VarType>_<nElements>_2
3 (XYVA)	ARRAY OF SMC_CAMXYVA

nTappets: INT

Number of switch actions.

pt: POINTER TO SMC_CAMTappet

Pointer on an ARRAY OF SMC_CAMTappet.

strCAMName:STRING

Name of the CAM.

SMC_CAMXYVA

A XYVA-CAM consists of an array of SMC_CAMXYVA. Each variable of that array describes a point of the CAM via **dX** (master position), **dY** (slave position), **dV** (first derivative dY/dX; corresponds to the slave velocity at a constant master velocitiy 1) and **dA** (second derivative d^2Y/dX^2; corresponds the slave-acceleration at a constant master velocity 1). Start- and end point of the CAM must be contained at least.

SMC_CAMTable_<variables-type>_<number of elements>_1

In this data structure an equidistant curve table is described. The particular slave positions are stored in **Table: ARRAY[0..<Anzahl Elemente>-1] OF <Variablen-Typ**. The start and end points of the CAM must be contained at least.

The variables **fEditorMasterMin**, **fEditorMasterMax**, **fTableMasterMin**, **fTableMasterMax** descirbe an additional scaling of the tables by storing the range of definitions/values in SoftMotion units (fEditorMaster, fEditorSlave) and scaled on table units (fTableMaster, fTableSlave).

SMC_CAMTable_<variable-type>_<number of elements>_2

A non-equidistant curve table is stored in **Table: ARRAY[0..<number of elements>-1] OF ARRAY[0..1] OF <variables-type>**. Contrary to the equidistant form the first element is the master position, the second one the slave position.

4.5.1 Example for a manually created CAM

This example shows how a CAM is created in the IEC program, i.e. without using the editor:

declaration part:

```
CAM: MC_CAM_REF:=(
     byType:=2, (* non-equidistant )
     byVarType:=2, (* UINT *)
     nElements:=128,
     xStart:=0,
     xEnd:=360);

Table: SMC_CAMTable_UINT_128_1:=(
     fEditorMasterMin := 0, fEditorMasterMax := 360,
     fTableMasterMin := 0, fTableMasterMax := 65536,
     fEditorSlaveMin := 0, fEditorSlaveMax := 360,
     fTableSlaveMin := 0, fTableSlaveMax := 65536);
```

program part:

```
(* Create CAM (a line for example); unique *)
FOR i:=0 TO 127 DO
     Table.Table[i][0]:=Table.Table[i][1]:=REAL_TO_UINT(i / 127.0 * 65536);
END_FOR

(* connect pointers; must be done in each cycle !!! *)
CAM.pce := ADR(Table);
```

The CAM created in this way now can be used as input for module MC_CamTableSelect and the output of this module in turn can be used for MC_CamIn.

5 The Library SM_PLCopen.lib

5.1 Overview

The modules provided by the library "SM_PLCopen.lib" follow the PLCopen specification: "Function blocks for motion control, Version 1.0".

This description is based on this specification and treats the **functionalities which are not covered by the PLCopen**.

The function blocks which are completely programmed according to the IEC1131-3 standard, can be classified in three categories:

1. Modules for the general operating, control and parameterizing of single drives. See a description in chapter 5.3.
2. Modules for the independent motion control of single drives. These modules help to move single axes autonomously in different ways.
3. Modules for the motion control or a drive (slave) against a further drive (master). These modules enable realizing CAMs, electronic gears and phase shifts. See a description in chapter 5.4.

Additional modules: see chapter 5.5.

Besides that for all important modules the library provides visualization templates, which are linked to an instance of the corresponding module and visualize its in- and outputs. These visualizations may be very useful during the programming and testing of an application.

Preconditions:

This library is basing on the library "Drive_Basic.lib ". This library provides the structure AXIS_REF, which is accessed by the library modules.

5.2 PLCopen-Specification "Function blocks for motion control, Version 1.0"

It is recommended to read - besides the on hand description - also the PLCopen-Specification "Function blocks for motion control, Version 1.0" . The main items are summarized in short in the following:

Modules get activated in two ways:

a) **Enable-Input**: If the module has an enable-input (like e.g. MC_ReadParameter), it will be active exactly as long as Enable is TRUE)

b) **Execute-Input**: The module gets activated by a rising edge (Transition from FALSE to TRUE) of the Execute-Input and not will get active again until it has been terminated its movement, or another module has taken control on the axis (AXIS_REF), or it has got a new rising edge at the Execute-Input thus re-starting the movement. Please also regard that all input variables only will be read in case of a rising edge.

By the Done-Output or another logical output the modules indicate either the validity of the outputs (e.g.MC_ReadStatus) or the termination of the movement (e.g. MC_MoveAbsolute).

A movement-creating module which gets interrupted by an other one, will indicate this by setting its CommandAborted-output.

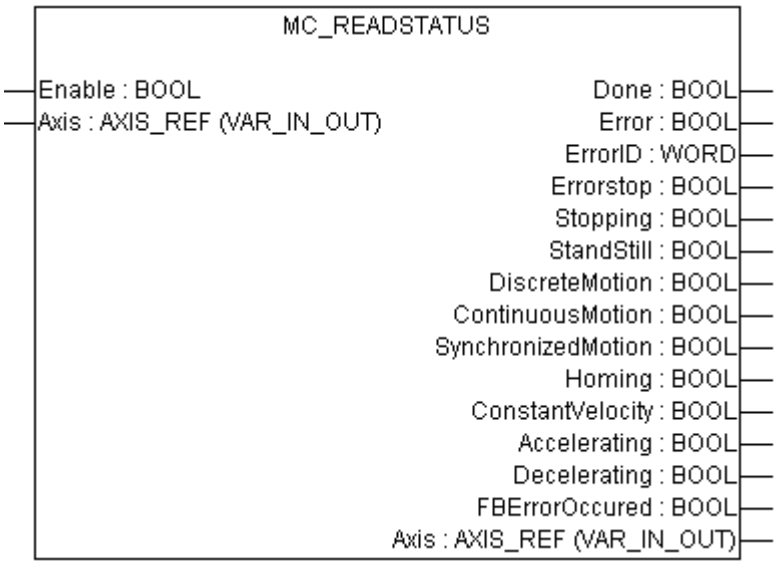
The outputs of the Execute-started modules - after having set their Done-Output - remain unchanged as long as the Execute-Input is set. By a falling edge they get deleted. If a falling edge has been detected before termination, the outputs will be set for one cycle and in the succeeding cycle will be deleted.

All motion generating modules require that in the corresponding axis the controller enable is done and the brake is released. Otherwise an error will be reported.

5.3 Modules for Controlling Single-Axis Motions

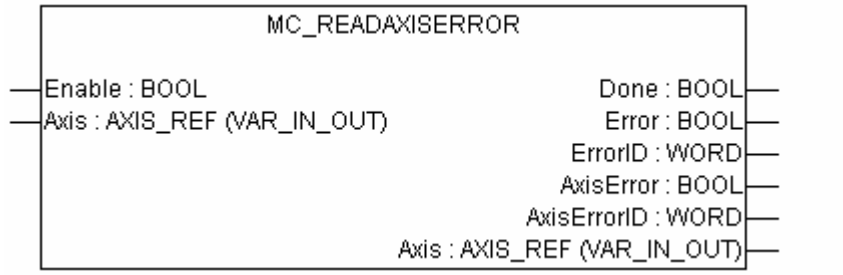
MC_ReadStatus

This function block, which is part of the SM_PLCopen.lib, provides some particular states of an axis.



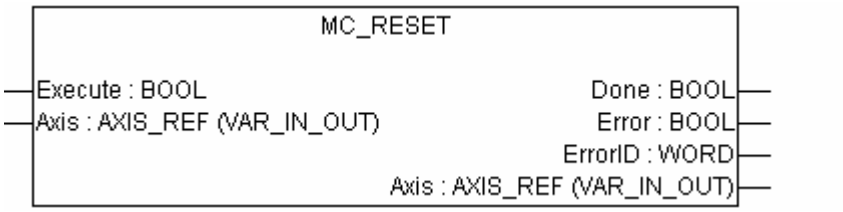
MC_ReadAxisError

This function block, which is part of the SM_PLCopen.lib, provides information on general errors which have occurred at the drive.



MC_Reset

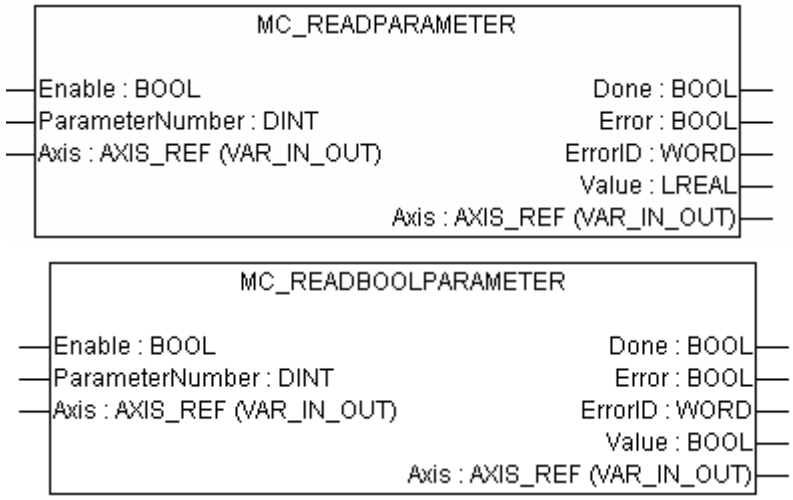
This function block, which is part of the SM_PLCopen.lib, reset the axis-state (SMC_AXIS_STATE) from "error_stop" to "standstill".



MC_ReadParameter,
MC_ReadBoolParameter

These function blocks, which are part of the SM_PLCopen.lib, can be used to read some standard parameters of the drive structure. Their numbers partially are specified by PLCopen, partially they are defined by the 3S – Smart Software Solutions GmbH Drive Interface.

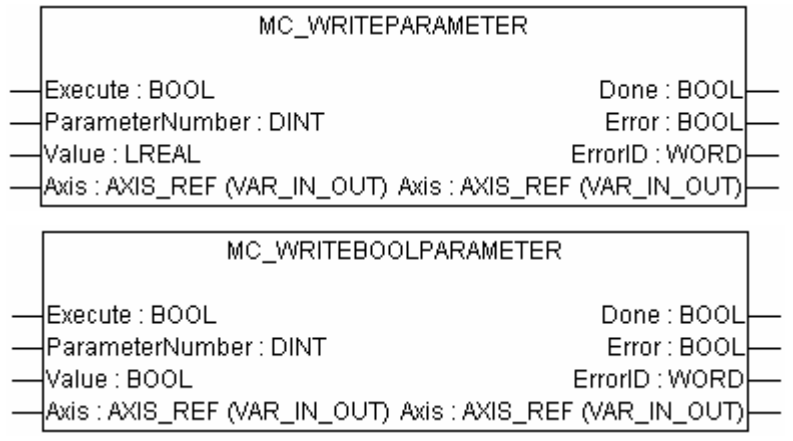
These modules also can be used to read manufacturer-specific data from the drive:. This modules also can be used to read manufacturer specific data from the drive. A document belonging to the particular drive library (XXXDrive.lib) will describe the coding of the drive specific parameter numbers.



MC_WriteParameter,
MC_WriteBoolParameter

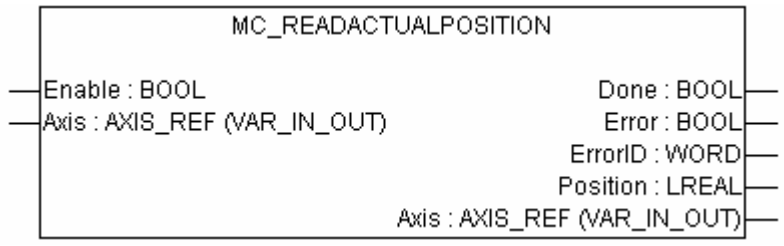
These function blocks, which are part of the SM_PLCopen.lib, can be used to set some standard parameters of the drive structure. Their numbers partially are specified by PLCopen, partially they are defined by the 3S – Smart Software Solutions GmbH Drive Interface.

These modules also can be used to send manufacturer-specific data to the drive: In this case they must get passed the negative (!) drive-specific parameter number. A document belonging to the particular drive library (XXXDrive.lib) will describe the coding of the drive specific parameter numbers.



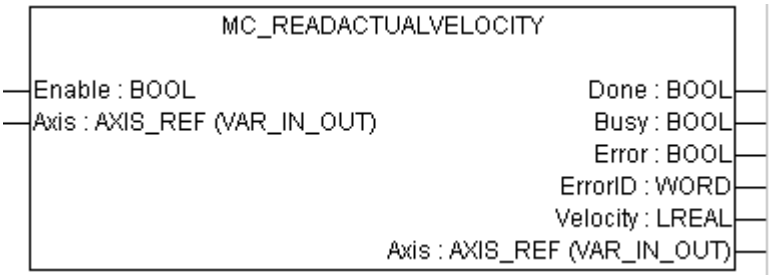
MC_ReadActualPosition

This function block, which is part of the SM_PLCopen.lib, provides the current position of the drive.



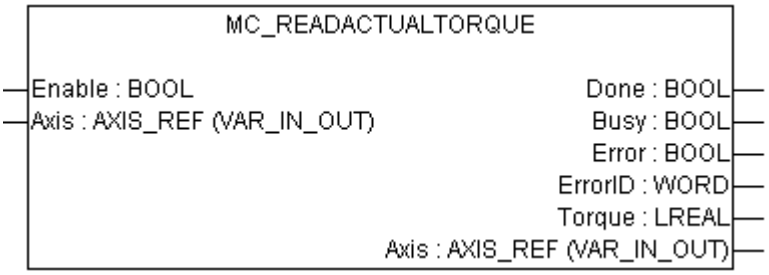
MC_ReadActualVelocity

This function block, which is part of the SM_PLCopen.lib, provides the current velocity of the drive.



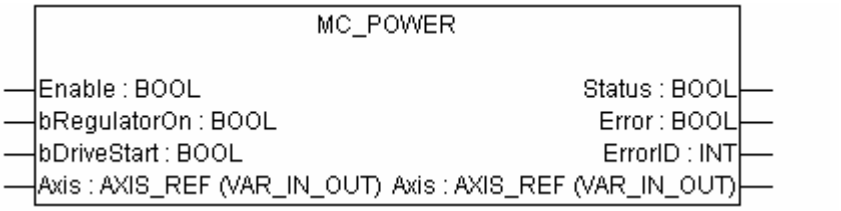
MC_ReadActualTorque

This function block, which is part of the SM_PLCopen.lib provides the current torque resp. the current power of the drive.



MC_Power

This function block, which is part of the SM_PLCopen.lib, controls the ON-/OFF-switch (power) and the status of the brakes of the drive. If a drive has not been switched ON in this way, if the controller has not been unblocked or if the brake has not been released, no motion control is possible.



Inputs of the function block:

Enable : BOOL (Default: FALSE)

As long as this variable is TRUE, the drive is switched on.

bRegulatorOn : BOOL (Default: FALSE)

Switches on/off the regulation.

bDriveStart : BOOL (Default: FALSE)

Applies resp. releases the brake in the drive.

In-/Output (VAR_IN_OUT) of the function block:

Axis : AXIS_REF

Here the structure is passed, which has been filled in the Drive Interface (Drive_Basic.lib) with the axis data.

Outputs of the function block:

Status : BOOL (Default: FALSE)

Indicates whether the drive currently is in (TRUE) or out (FALSE) of regulation.

Error : BOOL (Default: FALSE)

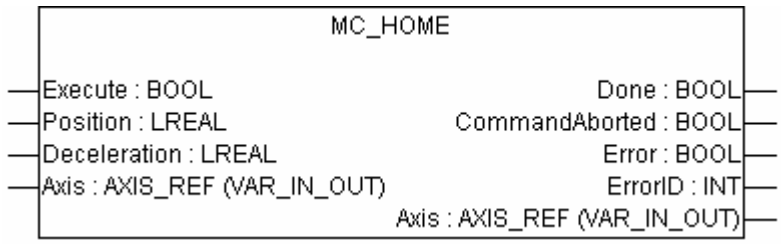
TRUE indicates an error in the function block.

ErrorID : INT;

Error number

MC_Home

This function block, which is part of the SM_PLCopen.lib, starts a manufacturer-specific (!) reference move in the drive. This motion is solely initiated by the Drive Interface. As soon as the drive signals that it has been finished, the output 'Done' will be set to TRUE.



Inputs of the function block:

Execute : BOOL (Default: FALSE)

At a rising edge at this variable the motion of the drive will be started.

Position : REAL

Absolute position of the drive when the reference signal is detected

Outputs of the function block:

Done : BOOL (Default: FALSE)

If TRUE, the reference move has been terminated and the drive is in standstill state.

CommandAborted : BOOL (Default: FALSE).

This variable gets TRUE if the command gets aborted by another.

Error : BOOL (Default: FALSE)

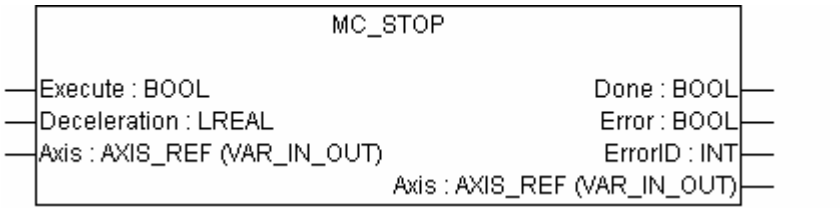
This variable gets TRUE when an error has occurred in the function block.

ErrorID : INT

Error number

MC_Stop

This function block, which is part of the SM_PLCopen.lib decelerates the axis to velocity 0. No interrupt is possible and the axis will be blocked as long as the input "Execute" is TRUE and the axis not yet has been stopped completely.



In-/Output (VAR_IN_OUT) of the function block:

Axis : AXIS_REF

This variable passes the structure, which has been feded with the axis data by the Drive Interface (Drive_Basic.lib).

Inputs of the function block:

Execute : BOOL (Default: FALSE)

At a rising edge at this variable the module will get active, this means will start the deceleration.

Deceleration : REAL

Value of the deceleration (decreasing energy of the motor) [u/s^2]

Outputs of the function block:

Done : BOOL (Default: FALSE)

This variable gets TRUE as soon as the commanded position has been reached, this means as soon as the drive has been stopped.

Error : BOOL (Default: FALSE)

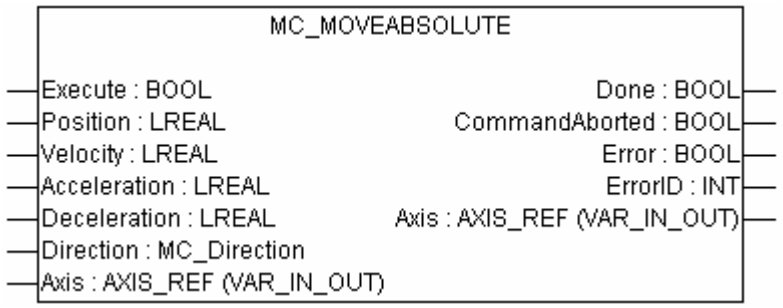
This variable gets TRUE when an error has occurred in the function block.

ErrorID : INT

Error number

MC_MoveAbsolute

This function block, which is part of the SM_PLCopen.lib, moves the axis to an absolute position according to the defined velocity, deceleration and acceleration values. In case of linear axes the direction value is not regarded, in case of rotating axes it determines the direction of rotation.



In-/Output (VAR_IN_OUT) of the function block:

Axis : AXIS_REF

This variable passes the structure, which has been feded with the axis data by the Drive Interface (Drive_Basic.lib).

Inputs of the function block:

Execute : BOOL (Default: FALSE)

At a rising edge at this variable the module will start the motion.

Position : REAL

Target position for the motion (technical unit [u])

Velocity : REAL

Value of the target velocity (not necessarily to be reached) [u/s] *)

Acceleration : REAL

Desired acceleration (increasing energy of the motor) [u/s^2]

Deceleration : REAL

Desired deceleration (decreasing energy of the motor) [u/s^2]

nDirection : MC_Direction (Default: shortest)

This enumeration provides the desired direction; only relevant for rotating axes (modulo-axis); see Drive_Basic.lib. Permissible values: current (current direction), positive, negative, shortest (seen from the current position), fastest (direction, which would finish movement as fast as possible) .

Outputs of the function block:

Done : BOOL (Default: FALSE)

This variable gets TRUE as soon as the commanded position has been reached, this means as soon as the drive has been stopped.

CommandAborted : BOOL (Default: FALSE)

This variable gets TRUE as soon as the commanded motion was interrupted by any motion function block acting on the same axis; except MoveSuperImposed

Error : BOOL (Default: FALSE)

This variable gets TRUE when an error has occurred in the function block.

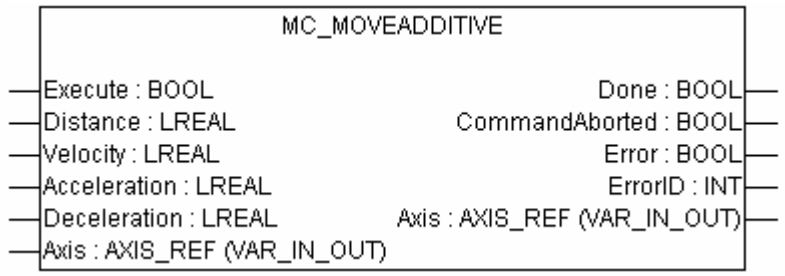
ErrorID : INT

Error number

MC_MoveAdditive

This function block, which is part of the SM_PLCopen.lib, has two different modes of action, depending on the current state of the axis:

1. discrete_motion:
   The Distance value will be added to the target position of the module which is currently processing on the axis. The motion will aim at the new target position then.
2. continuous_motion or standstill:
   The Distance will be covered referring to the current position, regarding the given parameters.



In-/Output (VAR_IN_OUT) of the function block:

Axis : AXIS_REF

This variable passes the structure, which has been feded with the axis data by the Drive Interface (Drive_Basic.lib).

Inputs of the function block:

Execute : BOOL (Default: FALSE)

At a rising edge at this variable the module will start the motion.

Distance : REAL

Relative distance for the motion (in technical unit [u]).

Velocity : REAL

Value of the target velocity (not necessarily to be reached) [u/s].

Acceleration : REAL

Desired acceleration (increasing energy of the motor) [u/s^2].

Deceleration : REAL

Desired deceleration (decreasing energy of the motor) [u/s^2].

Outputs of the function block:

Done : BOOL (Default: FALSE)

This variable gets TRUE as soon as the commanded position has been reached, this means as soon as the drive has been stopped.

CommandAborted : BOOL (Default: FALSE)

This variable gets TRUE as soon as the commanded motion was interrupted by any motion function block acting on the same axis; except MoveSuperImposed.

Error : BOOL (Default: FALSE)

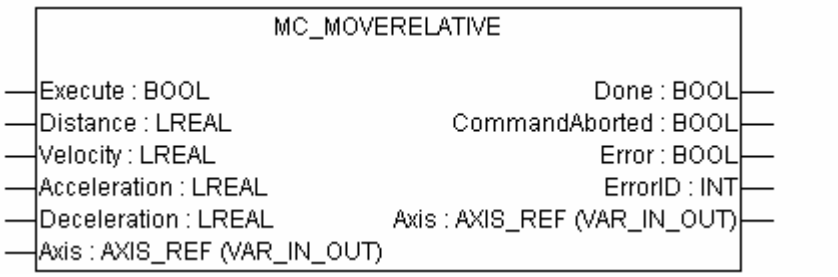
This variable gets TRUE when an error has occurred in the function block.

ErrorID : INT

Error number

MC_MoveRelative

This function block, which is part of the SM_PLCopen.lib, moves the axis by a relative distance according to the defined velocity, deceleration and acceleration values. The distance can have positive or negative values.



In-/Output (VAR_IN_OUT) of the function block:

Axis : AXIS_REF

This variable passes the structure, which has been feded with the axis data by the Drive Interface (Drive_Basic.lib).

Inputs of the function block:

Execute : BOOL (Default: FALSE)

At a rising edge at this variable the module will start the motion.

Distance : REAL

Relative distance for the motion (in technical unit [u]).

Velocity : REAL

Value of the target velocity (not necessarily to be reached) [u/s].

Acceleration : REAL

Desired acceleration (increasing energy of the motor) [u/s^2].

Deceleration : REAL

Desired deceleration (decreasing energy of the motor) [u/s^2].

Outputs of the function block:

Done : BOOL (Default: FALSE)

This variable gets TRUE as soon as the commanded position has been reached.

CommandAborted : BOOL (Default: FALSE)

This variable gets TRUE as soon as the commanded motion was interrupted by any motion function block acting on the same axis; except MoveSuperImposed

Error : BOOL (Default: FALSE)

This variable gets TRUE when an error has occurred in the function block.

ErrorID : INT

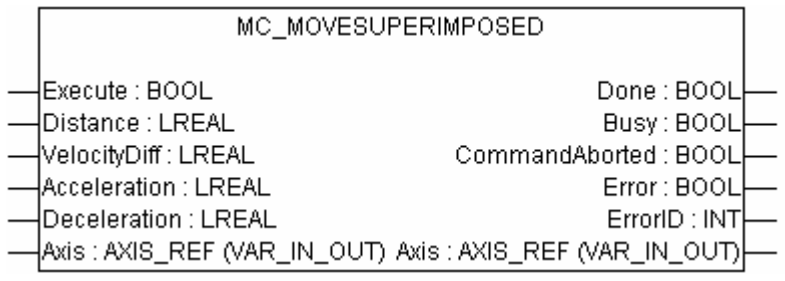
Error number

MC_MoveSuperImposed

This function block, which is part of the SM_PLCopen.lib causes, where appropriate, additionally to the currently active motion another one, which makes the axis passing a defined distance. The given values for velocity, acceleration and deceleration must be regarded as relative values, this means that they are independent of the underlying motion. The originally active module will not be interrupted by MC_MoveSuperImposed. If the originally active module gets interrupted by another module while MC_MoveSuperImposed still is active, MC_MoveSuperImposed nevertheless will continue the started motion, additionally to activity of the new module.

The basically active module will not be interrupted by MC_MoveSuperImposed. If the basically active module gets interrupted by another one, while MC_MoveSuperImposed is active, then the movement of MC_MoveSuperImposed will be aborted.

Please regard, that MC_MoveSuperImposed may not be called before the module which creates the underlying motion !



In-/Output (VAR_IN_OUT) of the function block:

Axis : AXIS_REF

This variable passes the structure, which has been feded with the axis data by the Drive Interface (Drive_Basic.lib).

Inputs of the function block:

Execute : BOOL (Default: FALSE)

At a rising edge at this variable the module will start the motion.

Distance : REAL

Relative distance for the motion (in technical unit [u]).

VelocityDiff : REAL

Value of the maximum velocity difference to the ongoing motion (not necessarily reached) [u/s].

Acceleration : REAL

Desired acceleration (increasing energy of the motor) [u/s^2].

Deceleration : REAL

Desired deceleration (decreasing energy of the motor) [u/s^2].

Outputs of the function blocke:

Done : BOOL (Default: FALSE)

This variable gets TRUE as soon as the commanded position has been reached.

Busy : BOOL (Default: FALSE)

This variable is TRUE as long as the superimposed motion currently is being processed.

CommandAborted : BOOL (Default: FALSE)

This variable gets TRUE as soon as the commanded motion was interrupted by any motion function block acting on the same axis.

Error : BOOL (Default: FALSE)

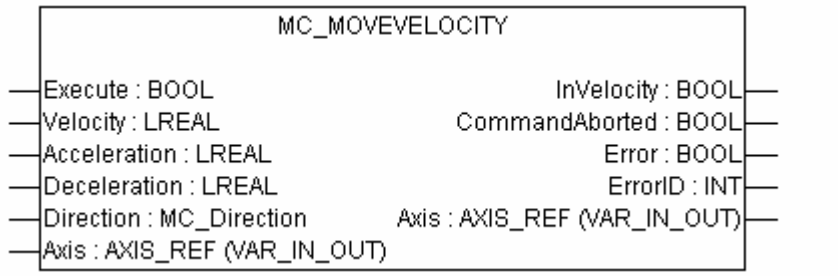
This variable gets TRUE when an error has occurred in the function block.

ErrorID : INT

Error number

MC_MoveVelocity

This function block, which is part of the SM_PLCopen.lib causes a non-stop motion of the axis with a predefined velocity (AXIS_REF). In order to reach this velocity, MC_MoveVelocity uses the programmed acceleration and deceleration values. The target velocity always is positive. The input variable *nDirection* defines the direction.



In-/Output (VAR_IN_OUT) of the function block:

Axis : AXIS_REF

This variable passes the structure, which has been fed with the axis data by the Drive Interface (Drive_Basic.lib).

Inputs of the function block:

Execute : BOOL (Default: FALSE)

At a rising edge at this variable the module will start the motion.

Velocity : REAL

Value of the maximum velocity difference to the ongoing motion (not necessarily reached) [u/s].

Acceleration : REAL

Desired acceleration (increasing energy of the motor) [u/s^2].

Deceleration : REAL

Desired deceleration (decreasing energy of the motor) [u/s^2].

Direction : MC_Direction (Default: shortest)

This enumeration provides the desired direction; only relevant for rotating axes (modulo-axis); see Drive_Basic.lib. Permissible values: current, positive, negative.

Outputs of the function block:

InVelocity : BOOL (Default: FALSE)

This variable gets TRUE as soon as the set velocity has been reached.

CommandAborted : BOOL (Default: FALSE)

This variable gets TRUE as soon as the commanded motion was interrupted by any motion function block acting on the same axis; except MoveSuperImposed.

Error : BOOL (Default: FALSE)

This variable gets TRUE when an error has occurred in the function block.

ErrorID : INT

Error number

MC_PositionProfile

This function block, which is part of the SM_PLCopen.lib, follows a defined position profile. For this purpose a variable of type structure MC_TP_REF must be defined and filled.

MC_TP_REF contains the following variables:

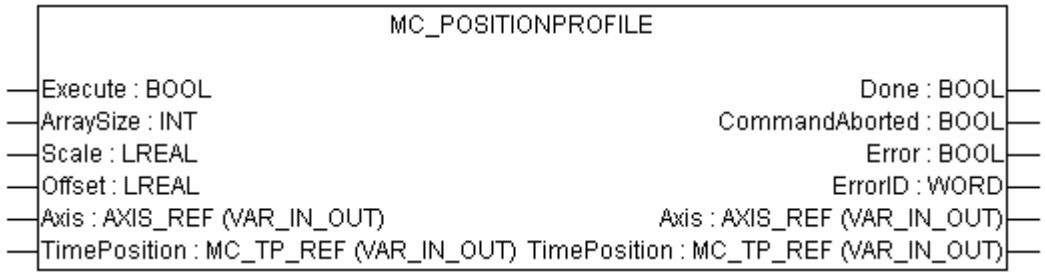
Variable	Type	Init value	Description
Number_of_pairs	INT	0	Number of profile position points
IsAbsolute	BOOL	TRUE	Positions absolute or relative
MC_TP_Array	ARRAY[1..100] OF MC_TP		Position points

MC_TP contains the following variables:

Variable	Type	Init value	Description
delta_time	TIME	t#0s	Period of time between reaching the last and the current position point
position	REAL	0	(Absolute-/relative-) position of the profile position point

The module creates a path through the given position points, which is a double continuously differentiable curve composed of cubic polynomials.

Regard, that the axis normally will reach the end of the predefined profile with a velocity and acceleration unequal 0. This can be compensated by the call of a MC_MoveAbsolute or a SMC_Stop module subsequently to the module MC_PositionProfile, after this has terminated its work.



In-/Output (VAR_IN_OUT) of the function block:

Axis : AXIS_REF

Here the structure is passed, which has been filled in the Drive Interface (Drive_Basic.lib) with the axis data.

Inputs of the function block:

Execute : BOOL (Default: FALSE)

At a rising edge at this variable the module will start the motion.

ArraySize : INT

Dimension of array for position points (max. 1..100).

Scale : REAL (Default: 1)

General scaling factor of the profile.

Offset : REAL

General offset of profile [u].

TimePosition : MC_TP_REF

Information on time and position values, see above.

Outputs of the function block:

Done : BOOL (Default: FALSE)

This variable gets TRUE as soon as the commanded position has been reached.

CommandAborted : BOOL (Default: FALSE)

This variable gets TRUE as soon as the commanded motion was interrupted by any motion function block acting on the same axis; except MoveSuperImposed.

Error : BOOL (Default: FALSE)

This variable gets TRUE when an error has occurred in the function block.

ErrorID : INT

Error number

MC_VelocityProfile

This function block, which is part of the SM_PLCopen.lib, is an analog to the module MC_PositionProfile. But here in the input variable of type structure MC_TV_REF the position points are defined by their velocities.

MC_TV_REF contains the following variables:

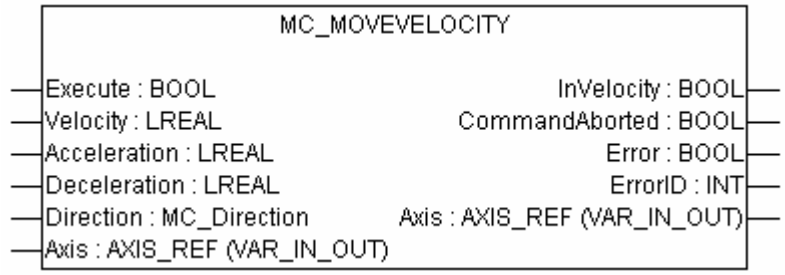
Variable	Typ	Init value	Description
Number_of_pairs	INT	0	Number of profile position points
IsAbsolute	BOOL	TRUE	Positions absolute or relative
MC_TV_REF	ARRAY[1..100] OF MC_TV		Position points

SMC_TV_REF contains the following variables:

Variable	Type	Init value	Description
delta_time	TIME	t#0s	Period of time between reaching the last and the current position point
velocity	REAL	0	(Absolute-/relative-)velocity of the position point

The module creates a path through the given position points, which is a continuously differentiable curve consisting of parables.

The position of the axis results from the start position and the integrated velocity.



MC_AccelerationProfile

This function block, which is part of the SM_PLCopen.lib, is an analog to the MC_PositionProfile module. But here in the input variable of type structure MC_TA_REF the position points are defined by their acceleration values.

MC_TA_REF contains the following variables:

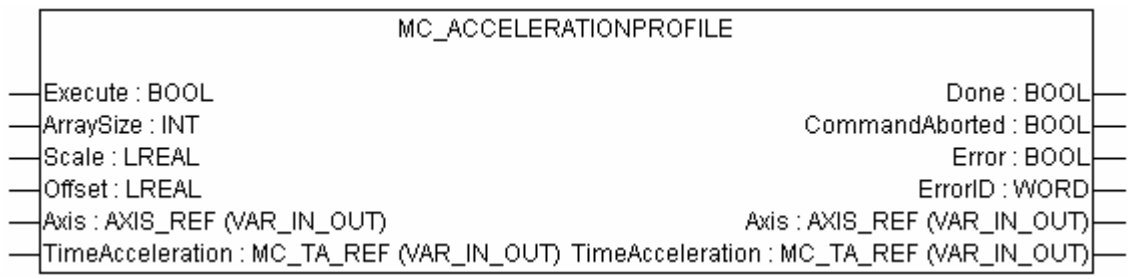
Variable	Type	Init value	Description
Number_of_pairs	INT	0	Number of profile position points
IsAbsolute	BOOL	TRUE	Positions absolute or relative
MC_TA_Array	ARRAY[1..100] OF MC_TA		Position points

MC_TA contains the following variables:

Variable	Type	Init value	Description
delta_time	TIME	t#0s	Period of time between reaching the last and the current position point
acceleration	REAL	0	(Absolute-/relative-)velocity of the position point

The module creates a path through the given position points, which is a continuous curve consisting of lines.

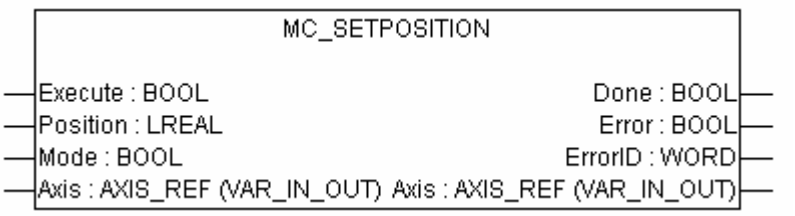
The velocity of the curve results from the velocity at the start of the profile and the and the integrated acceleration. The position of the axis results from the start position and the integrated velocity.



MC_SetPosition

This module shifts the zero point of the axis, so that:

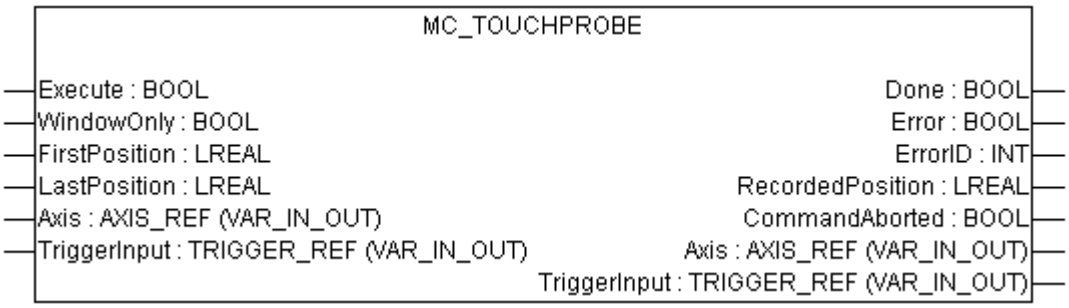
- in absolute mode (Mode = FALSE; Default) the value which is set by input *Position* will become the actual set position, resp.
- in relative mode (Mode = TRUE) the actual set position will be shifted by the size of *Position*.



Basically the module can be called at any time. But regard that at a path-controlled motion, if the target positions are given to the module directly, e.g. via SMC_ControlAxisByPos, a jump of the target position break can result.

MC_TouchProbe

This function block, which is part of the SM_PLCopen.lib can be used to detect very precisely the position of the drive via a fast input. Because this as a rule must work faster than in the normal PLC, in many cases either the drive is impinged with this function or it is executed - independently from the PLC cycles - via interrupts or the like.



Input TriggerInput is of type TRIGGER_REF and describes the trigger input in detail:

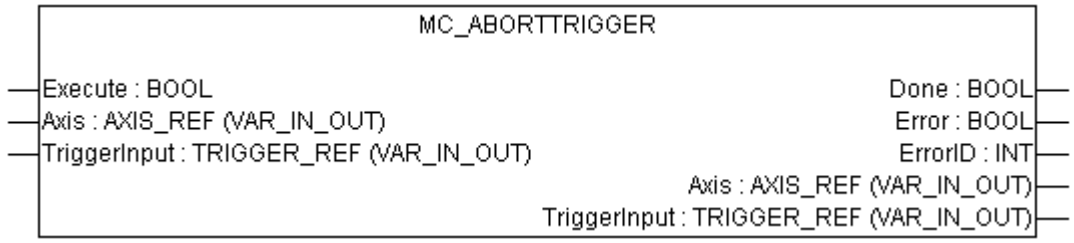
Variable	Type	Initial value	Description
bFastLatching	BOOL	TRUE	quick latching via DriveInterface (TRUE) or latching according to PLC cycle (FALSE)
iTriggerNumber	INT	-1	only for bFastLatching = TRUE: Trigger number; depending on DriveInterface
bInput	BOOL	FALSE	only for bFastLatching = TRUE::Input signal; TRUE causes latching.
bActive	BOOL	FALSE	internal variable

The window function, activated and defined via WindowOnly, FirstPosition, LastPosition, is dependent on being supported by the DriveInterface and will return an error if this is not the case.

The module is independent from the axis state and is active until a position will be latched resp. the process will be aborted by MC_AbortTrigger.

MC_AbortTrigger

This function block, which is part of the SM_PLCopen.lib aborts a latching which is currently done on the trigger input.



5.4 Modules for Synchronized Motion Control

MC_CamTableSelect

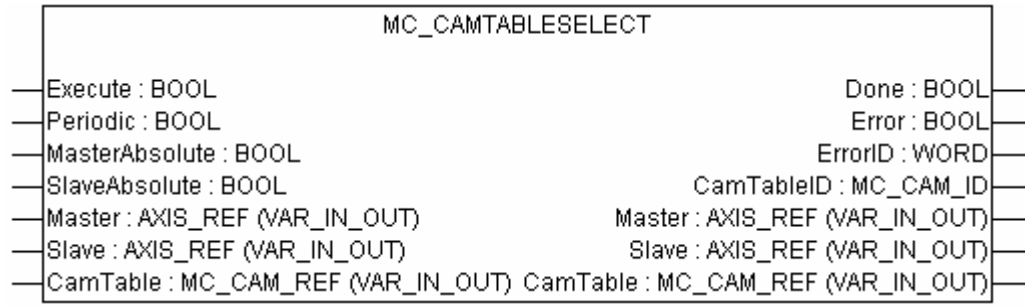
Using this module, which is provided by the library SM_PLCopen.lib, you can select a CAM, determine the master and the slave axes for this CAM and do some pre-settings. The object *CamTableID* which is available as an output, later will be passed on to the CAM module MC_CamIn.

The master axis can be virtual, this means that it must not exist physically. If the variable *bRegulatorOn* is TRUE, the target values of the master axis will be used, otherwise the actual values.

The CAM defining the motion either can be programmed manually in an structure object of type MC_CAM_REF or it can be created in the CAM editor which is integrated in the CoDeSys programming system. (see document "SoftMotion CAM-Editor").

If the variable *Periodic* is TRUE, after a complete pass the processing of the CAM will be restarted, otherwise just one run will be done.

The variables *MasterAbsolute* und *SlaveAbsolute* define, whether the CAM-mapping of the master axis to the slave axis should refer to absolute values (TRUE) or to increments (FALSE).



In-/Outputs (VAR_IN_OUT) of the module:

Master : AXIS_REF

Master axis

Slave : AXIS_REF

Slave axis

CamTable : MC_CAM_REF

Description of the CAM

Inputs of the module:

Execute : BOOL (Default: FALSE)

At a rising edge the module chooses a new CAM.

Periodic : BOOL (Default: TRUE)

periodic/ non-periodic CAM.

MasterAbsolute : BOOL (Default: TRUE)

CAM refers to absolute/ relative (referring to position at rising edge in Execute of CAMIn) master position.

SlaveAbsolute : BOOL (Default: TRUE)

CAM refers to absolute/ relative (referring to position at rising edge in Execute of CAMIn) slave position.

Outputs of the module:

Done : BOOL (Default: FALSE)

TRUE indicates that the desired distance has been covered

Error : BOOL (Default: FALSE)

TRUE indicates that an error has occurred in the function block

ErrorID : SMC_Error (INT)

Error number

CAMTableID : MC_CAM_ID

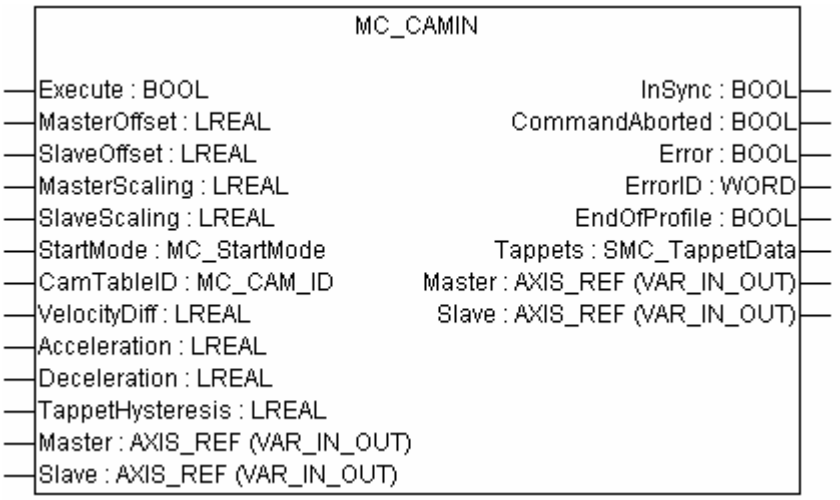
Output describing the CAM. Serves as an input for the homonymous input in MC_CamIn.

MC_CamIn

Using this module, which is provided by the library SM_PLCopen.lib, you can realize a CAM which has been selected by MC_CAMTABLESELECT.

In addition to the offsets and scaling also the start mode can be defined. **Regard** that the modes *ramp_in*, *ramp_in*_pos and *ramp_in_neg*, which would effect a continuous approximation of the slave target value to the CAM target value, in case at start time the actual slave value would differ from the CAM target value, is not yet implemented.

This module provides an additional function. It detects tappets and via the output *Tappets* can hand over the tappet info to one or several *SMC_GetTappetValue* function blocks (see *SMC_GetTappetValue*). Regard that the *CamIn* Module cannot register more than three tappets per cycle. The module *SMC_CAMRegister* works without this limitation.



In-/Outputs (VAR_IN_OUT) of the module:

Master : AXIS_REF

Master axis

Slave : AXIS_REF

Slave axis

Inputs of the module:

Execute : BOOL (Default: FALSE)

At a rising edge the module starts the movement

MasterOffset : LREAL (Default: 0)

additional offset on master position

SlaveOffset : LREAL (Default: 0)

additional offset on slave position

StartMode : MC_StartMode (absolute/relative/ramp_in/ramp_in_pos/ramp_in_neg) (Default: absolute)

CAM either is started relative (relative) to the current position or absolutely (absolute) to this, or with slow ramping in (ramp_in), in positive (ramp_in_pos) or negative (ramp_in_neg) direction.

CamTableID : MC_CAM_ID

Output of MC_CamTableSelect

Velocity, Acceleration, Deceleration: LREAL (Default: 0)

additonal velocity, acceleration, deceleration for ramp_in mode

TappetHysteresis: LREAL (Default: 0)

Width of the hysteresis band around the tappets

Outputs of the module:

InSync : BOOL (Default: FALSE)

TRUE indicates that the movement is on the CAM

CommandAborted : BOOL (Default: FALSE)

The started movement has been aborted by another function block, which effects the same axis; Exception: MoveSuperImposed

Error : BOOL (Default: FALSE)

TRUE indicates an error in the function block

ErrorID : SMC_Error (INT)

Error number

EndOfProfile : BOOL

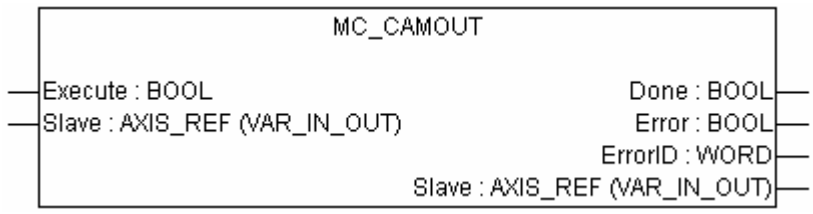
Indicates the end of a CAM. At periodic CAMS this output will be pulsed

Tappets : SMC_TappetData

Tappet output. The particular tappet positions finally will be evaluated by the SMC_GetTappetValue module.

MC_CamOut

Using this module, which is provided by the library SM_PLCopen.lib, you can disengage the slave drive from the master. The slave will be driven on with the current velocity.

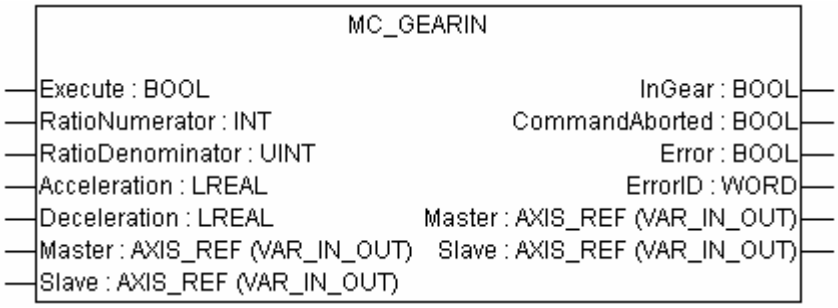


MC_GearIn

Using this module, which is provided by the library SM_PLCopen.lib, you can couple the slave axis to the master axis. Thereby the slave velocity is f-times the velocity of the master axis. The value of factor f results from the quotient of the input parameters *RatioNumerator* und *RatioDenominator*.

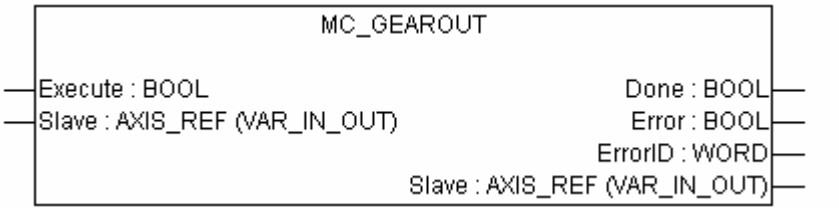
The module accelerates resp. decelerates the slave axis as long as its velocity will have the desired ratio, whereby the values of *Acceleration* and *Deceleration* will be regarded. As soon as this has been reached, the slave axis velocity derives from the master axis.

If the variable *bRegulatorOn* (structure AXIS_REF, Drive_Basic.lib) of the master axis is TRUE, the set values of the velocity will be used, otherwise the actual values.



MC_GearOut

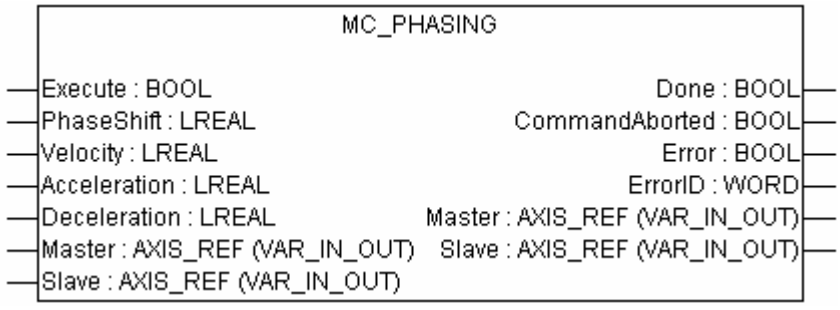
Using this module, which is provided by the library SM_PLCopen.lib, you can disengage the slave drive from the master. The slave will be driven on with the current velocity.



MC_Phasing

Using this module, which is provided by the library SM_PLCopen.lib, you can effect a constant distance between master axis and slave axis. In this case master and slave of course have identically velocity and acceleration. For this purpose the slave axis by acceleration or deceleration gets the same velocity as the master axis. When this state has been reached, on the master axis an additional movement will be executed (similarly to MC_MoveSuperImposed), which will effect the desired phase shift.

The MC_Phasing module will stay active until it will be interrupted by another Stein.



5.5 Additional Elements of the SM_PLCopen.lib

SMC_GetCamSlaveSetPosition

This module calculates the current target position of an axis (slave) for the case that the axis would be coupled via a CAM to the motion of another axis (Master). Thereby both axes are not moved or affected.

The module can be used if a slave axis prior to connecting to a CAM should be moved to the target position which has resulted by that.

Due to the fact that the module calculates the corresponding value within a cycle, a done-output is not needed.

In-/Outputs (VAR_IN_OUT) of the module:

CamTableID : MC_CAM_ID

CAM; Output of MC_CamTableSelect.

Master : AXIS_REF

Master axis.

Slave : AXIS_REF

Axis for which the CAM target position is calculated.

Inputs (VAR_IN) of the module (all inputs not described in the folllowing correspond to those of MC_CamIn):

Enable : BOOL

Activates the module.

Outputs of the module:

fStartPosition : LREAL

Calculated target position for the slave

Error : BOOL (Default: FALSE)

TRUE indicates that in the module an error has occured.

ErrorID : SMC_Error (INT)

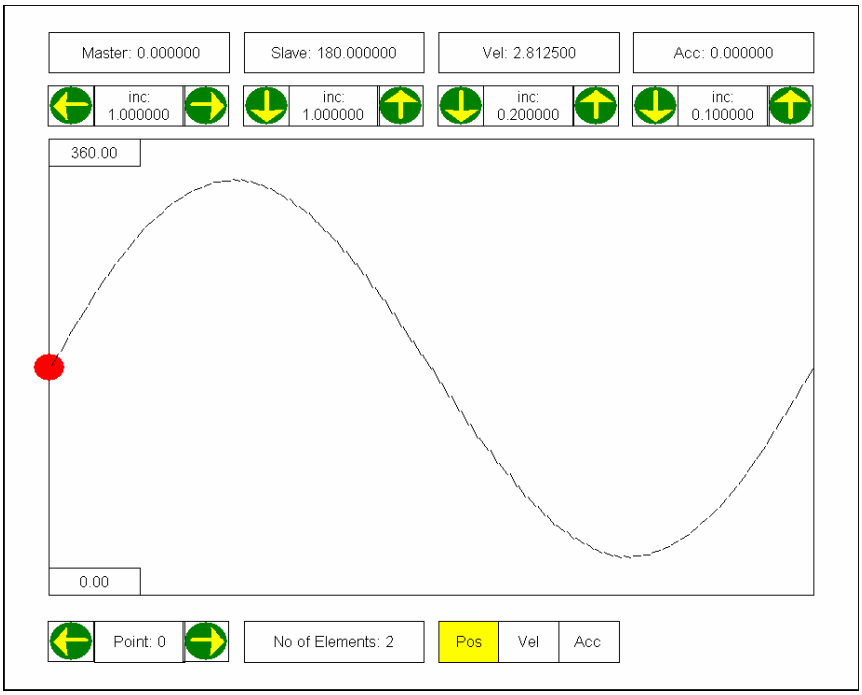
Error number

SMC_CAMEditor,
SMC_CAMVisu

With these modules an Online-CAM-Editor can be created.

SMC_CAMEditor must be called in the SoftMotion-task, whereas SMC_CAMVisu should be called in a slower task of lower priority.

Both modules should be connected with the corresponding visualization template (SMC_CAMEditor), which represents the given CAM and allows the user to modify that CAM also during run time.



The red circle marks the current CAM point. That can be changed via the arrows in the lower left corner. The buttons in the lower right can be used to select whether position, velocity or acceleration should be displayed. The arrows in the upper bar can be used to move the master-, slave-position, slave-velocity and –acceleration by the specific increment.

In-/Outputs (VAR_IN_OUT) of the module SMC_CAMEditor:

CAM : MC_CAM_REF

CAM to be visualized and modified.

Inputs of module SMC_CAMEditor:

Enable : BOOL (Default: FALSE)

Getting TRUE activates the module .

dYPeriod : LREAL

Slave period (for periodic CAMs).

bPeriodic : BOOL (Default: TRUE)

TRUE for periodic CAM, otherwise FALSE.

In-/Outputs (VAR_IN_OUT) of module SMC_CAMEditor:

ce:SMC_CAMEditor

SMC_CAMEditor-Instance.

SMC_CAMRegister

This function block represents a tappet control unit. It works - like MC_CamIn – on a MC_CAM_REF-structure, negating the original path information and only reading the tappet information.

In-/Outputs (VAR_IN_OUT) of the function block:

Master : AXIS_REF

Describes the axis structure (see Chapter 2.4) which should switch the tappets.

CamTable : MC_CAM_REF

Description of a (maybe empty) CAM containing the description of the tappets.

bTappet : ARRAY [1..MAX_NUM_TAPPETS] OF BOOL

Tappet bits.

Inputs of the function block:

Enable : BOOL (Default: FALSE)

At TRUE the function block starts to switch the tappets.

MasterOffset : REAL

Offset to be added to the master position.

MasterScaling : REAL (Default: 1)

General scale factor for master axis.

TappetHysteresis : REAL

Hysteresis around tappets.

DeadTimeCompensation : REAL

Dead time in sec. The position of the master axis to be expected will be calculated by linear extrapolation.

Outputs of the function block:

Error : BOOL (Default: FALSE)

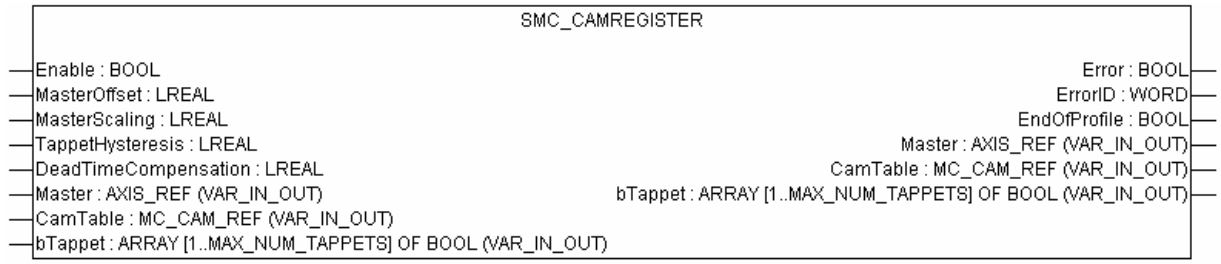
TRUE indicates that an error has occurred in the function block.

ErrorID : INT

Error number

EndOfProfile : BOOL

At the transition from the end of the path (profile) to the start this output gets true for the time period of one cycle



SMC_GetTappetValue

This function block evaluates the output *Tappets* of function block MC_CamIn and contains the current tappet status.

In-/Outputs (VAR_IN_OUT) of the function block:

Tappets : SMC_TappetData

Inputs of the function block:

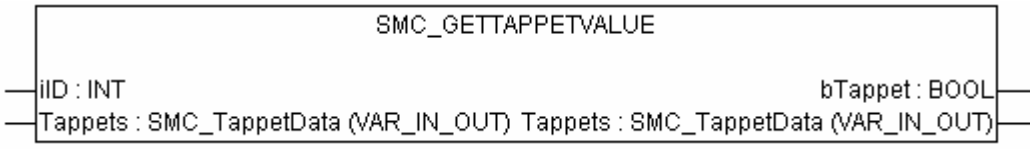
iID : INT

Group-ID of the tappet to be evaluated.

Outputs of the function block:

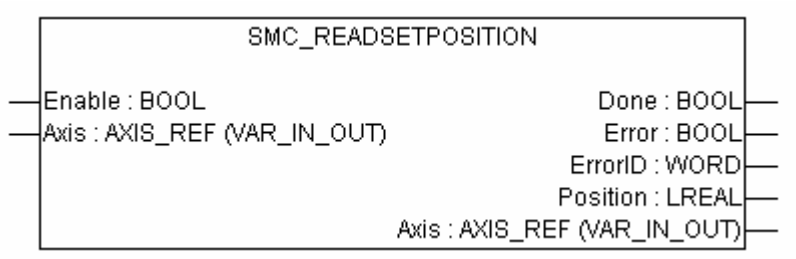
bTappet : BOOL (Default: FALSE)

Tappet value



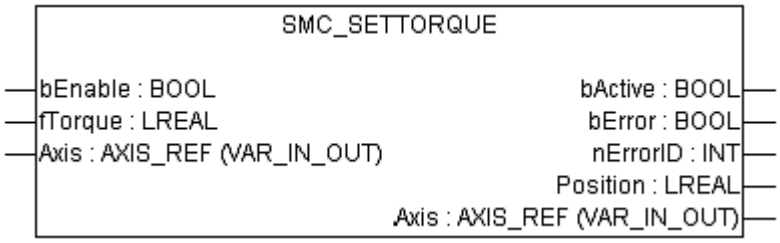
SMC_ReadSetPosition

This module reads the current set position of the drive.



SMC_SetTorque

This module can be used to create a torque, if the drive is in controller mode "torque".



6 The Library SM_CNC.lib

6.1 Overview

This library provides modules for realizing the SoftMotion functionality in an IEC-program. For this purpose it has to be included in the IEC-program. Among them there are modules and structures which on the one hand execute the movements, which have been designed in the editors, and on the other hand can realize profiles which get designed online by the IEC-program:

- SMC_NCDecoder Decoding of the path which has been programmed in the CNC-Editor in order to get structure objects
- SMC_ToolCorr Path-preprocessing: tool radius correction
- SMC_AvoidLoop Path-preprocessing: avoids loops in the path
- SMC_SmoothPath Path-preprocessing: slurs the path by splines
- SMC_RoundPath Path-preprocessing: rounds the path by circular arcs
- SMC_CheckVelocities Check of the end velocities of the segments
- SMC_Interpolator Conversion of the decoded, eventually preprocessed path objects to discrete points
- Auxiliary functions for moving and rotating a path
- Global variables Setting some internal parameters
- Structures SMC_POSINFO, SMC_GEOINFO, SMC_VECTOR3D and SMC_VECTOR6D Save of positions, path segments and vectors)
- Structure SMC_OUTQUEUE Managing GeoInfo-objects in a list of defined size

The modules and structures will be described in the following chapters. Also have a look at the programming examples in Chapter 11)

Note:	Regard that the CNC-Editor can compile a CNC program in two different ways: as program variable (SMC_CNC_REF), which must run through Decoder- and if applicable path-preprocessing modules, or as OUTQUEUE, which can be directly passed on to the Interpolator.

6.2 Modules

6.2.1 SMC_NCDecoder

The function block SMC_NCDecoder is used to convert a CNC program, which has been created in the CNC-Editor, to a list of SoftMotion-GEOINFO-structure objects (SM_CNC.lib). One line of the program is decoded per cycle.

Inputs of the module:

bExecute: BOOL

The function block will do a reset and start the decoding as soon as a rising edge is detected at this input.

bAppend: BOOL

As long as this input is FALSE, at each reset the DataOutQueue will be cleared. As long as it is TRUE, newly incoming data will be written to the end of the DataOutQueue.

bStepSuppress: BOOL

If this input is TRUE (default), lines of the CNC program, starting with ‚/', will be ignored. At FALSE (default) they will be processed anyway.

piStartPosition

Position of the point to be moved at the beginning of the path.

nSizeOutQueue: UDINT

Size of the data buffer, to which the list of GEOINFO structure objects should be written. This buffer must be at least five times as big as a GEOINFO structure, this means about 2KB. If this is not the case, SMC_NCDecoder will not execute any actions at all. The value can be set but may only be modified later during a reset.

pbyBufferOutQueue: POINTER TO BYTE

This input must point to the first byte of the memory area which has been allocated for the OUTQUEUE-structure. This area must be at least as big as defined in nSizeOutQueue. Typically the allocation of the memory buffer is done in the declaration part of the IEC-program by defining a byte-array (e.g. `BUF: ARRAY[1..10000] OF BYTE;` for a 10000 byte memory area). This value can be predefined, but later it can only get modified during a reset.

In/Outputs (VAR_IN_OUT) of the module:

ncprog: BYTE

In this IN_OUT variable the CNC program (structure SM_CNC_REF, Drive_Basic.lib) will be passed on. This program may have been created by the IEC-program or in the CNC-Editor.

Outputs of the module:

bDone: BOOL

This variable is set to TRUE, as soon as the processing of the program has been finished. Thereafter the SMC_NCDecoder will not perform any actions until it gets a reset. If input bExecute is FALSE, bDone will be set back to FALSE.

bError: BOOL

In case of an error this input gets TRUE.

wErrorID: SMC_ ERROR (INT)

This enum output describes an error which might be occurred during decoding. After an error the processing will be stopped until a reset is done.

poqDataOut: POINTER TO SMC_OUTQUEUE

This variable points to a SMC_OUTQUEUE-structure, which manages the decoded SMC_GEOINFO-objects.

iStatus: SMC_DEC_STATUS (INT)

This enum-variable shows the current status of the module. Possible states:

WAIT_PROG	0	Program not yet found
READ_WORD	1	Word read
PROG_READ	2	End of program reached

iLineNumberDecoded: INT

This variable contains the line number (not the sentence number) of the last processed line.

iErr: SMC_DEC_ERROR (INT)

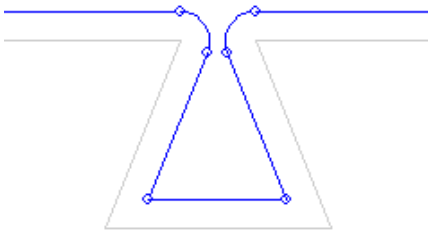
This enum-variable describes an error which may have occurred during decoding:

ERR_DEC_NO_ERROR	0	No error detected
ERR_DEC_ACC_TOO_LITTLE	1	Maximum acceleration is lower than one path-unit/sec^2
ERR_DEC_RET_TOO_LITTLE	2	Maximum delay is lower than one path-unit/sec^2

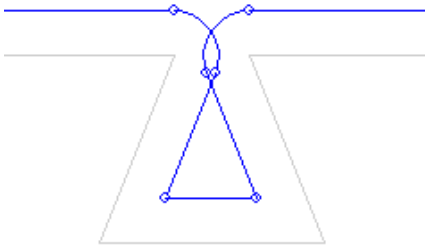
As soon as an error has occurred, the processing gets stopped until a reset will be done.

6.2.2 SMC_ToolCorr

The *SMC_ToolCorr*-module can be used for path-preprocessing. It creates a shifted path basing on the originally defined profile. In the shifted path each point of each path object has a definable distance to the original point and the immediate neighbour points (Tool radius correction). Thus the shifted path guarantees that each of its points has a **fix distance** to the original path. A typical application is the milling of a programmed contour with a milling drill of defined diameter. In order to compensate the radius of the drill the milling drill must follow an appropriately shifted path – which can be created by using the *SMC_ToolCorr* Module.



! The following restriction must be regarded: If the outline and the drilling radius are chosen in a manner that cross-over points would result within the shifted path – which would effect that the desired contour would be destroyed during passing the shifted path - then this will not be regarded (see the drawing to the right of this paragraph). To avoid such intersections, use the module SMC_AvoidLoop.



The *SMC_ToolCorr* module is working as described in the following:

All SMC_GEOINFO-objects which are found in the Input-OUTQUEUE-structure get checked one after the other. If in one of the objects Bit1 (Bit2) of the variable Intern_Mark is set, then starting from there – in direction of motion - the path will be shifted to the left (right) by the currently set tool radius. In order to get a continuous path, a positioning object (MoveType = 100) is inserted, resp. if such a positioning object is already preceding the object, it will be shifted directly to the start point of the shifted path. Each further object then will be shifted also, until Bit0 of Intern_Mark gets set. This will stop the tool radius correction. But also here a continuous prosecution of the path will be guaranteed by using a positioning object. A shift in the opposite direction only can be started if the currently started tool correction has been terminated before by setting Bit0. The SMC_NCDecoder will set these bits, thereby reacting on the instructions G41/G42/G40. In other words: **The tool radius correction will be done for all objects which are placed between the instructions G41 and G40 resp. G42 and G40**.

Inputs of the function block:

bExecute: BOOL

The function block will do a reset and start the tool correction as soon as a rising edge is detected at this input.

bAppend: BOOL

As long as this input is FALSE, at each reset the DataOutQueue will be cleared. As long as it is TRUE, newly incoming data will be written to the end of the DataOutQueue.

poqDataIn: POINTER TO SMC_OUTQUEUE

This is a pointer to the SMC_OUTQUEUE-structure object, which contains the SMC_GEOINFO-objects of the path to be shifted; typically it points on the output ..*DataOut* of the preceding module (e.g. the SMC_NCDecoder).

dToolRadius: LREAL

This variable contains the value, which determines – added to the current ToolRadius of the SMC_GEOINFO-object – the tool radius by which the path should be shifted (see above). This value can be modified online. Thus it is possible to predefine the value offline (by the SMC_GEOINFO-structure) and to modulate it online. Thereby regard that a tool radius correction which is initiated during the block is just being shifted, will cause an abort of the path correction and therefore should be avoided ! But it is possible to do the radius correction during a reset or in a phase where it is guaranteed that the module is not currently shifting a block (Status = TC_ORIG). Default: 0.

nSizeOutQueue: UDINT

This variable contains the size of the data buffer, to which the list of GEOINFO structure objects will be written. This buffer must be at least five times as big as a GEOINFO structure, this means about 2KB. If this is not the case, SMC_NCDecoder will not execute any actions at all. The value can be set but may only be modified later during a reset.

pbyBufferOutQueue: POINTER TO BYTE

This input variable must point to the first byte of the memory area which is allocated for the OUTQUEUE-structure. This area must be at least as big as defined in nSizeOutQueue. Typically the allocation of the memory buffer is done in the declaration part of the IEC-program by defining a byte-array (e.g. `BUF: ARRAY[1..10000] OF BYTE;` for a 10000 byte memory area). This value can be predefined, but later it only can be modified during a reset.

Outputs of the function block:

b_Done: BOOL

This variable will be set to TRUE as soon as the input data of DataIn have been processed completely. Thereafter the module will not perform any further actions until it gets a reset.

bError: BOOL

In case of an error this input gets TRUE.

wErrorID: SMC_ERROR (INT)

In case of an error this input shows the error number.

poqDataOut: POINTER TO SMC_OUTQUEUE

This variable points to a SMC_OUTQUEUE-structure, which manages the decoded SMC_GEOINFO-objects.

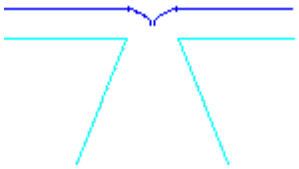
iStatus: SMC_TC_STATUS (INT)

This enum-variable shows the current status of the module. Possible states:

TC_ORIG	0	No tool radius correction at the object
TC_RIGHT	1	Shift objects to the right

TC_LEFT	2	Shift objects to the left
TC_END	4	Processing of the objects has been terminated

6.2.3 SMC_AvoidLoop



The *SMC_AvoidLoop* function block can be used for path-preprocessing. It creates a loopless path **copy** of a defined path. If in the original path an intersection is detected, the path will be cut at this point, the loop will be removed, and the path will be continued with the rest of the curve. Thus a loopless, continuous path results.

See for a typical application in the description of the SMC_ToolCorr module.

The *SMC_AvoidLoop* function block is working as described in the following:

The module passes all SMC_GEOINFO-objects which are found in the input-SMC_OUTQUEUE-structure. If in one of these objects Bit7 of the variable Intern_Mark is set, then the avoid-loop-functionality will be activated. It will start to check whether there is any intersection point of the current object with the subsequent SMC_GEOINFO-objects, which come before a SMC_GEOINFO-Object, in which Bit6 of the variable Intern_Mark is set. This bit will terminate the avoid-loop-functionality. If no intersections are found, the object will be copied unchanged to the Output-SMC_OUTQUEUE. Otherwise the first of the intersecting objects will be cut at the intersection point, the SMC_GEOINFO-objects positioned between the intersecting objects will be removed, and the new path will be continued with the second of the two intersecting objects. The SMC_NCDecoder will set the Bits 6 and 7 of Intern_Mark as an reaction on the instructions G61/G60.

Regard: It depends on the size Input-SMC_OUTQUEUE whether the SMC_AvoidLoop module can work correctly. If a loop contains more objects than can be stored in the SMC_OUTQUEUE, then the loop cannot get detected !

Inputs of the module:

bExecute: BOOL

The function block will do a reset and start the path correction (avoiding loops) as soon as a rising edge is detected at this input.

bAppend: BOOL

As long as this input is FALSE, at each reset the DataOutQueue will be cleared. As long as it is TRUE, newly incoming data will be written to the end of the DataOutQueue.

poqDataIn: POINTER TO SMC_OUTQUEUE

This variable points to the SMC_OUTQUEUE-structure object, which contains the SMC_GEOINFO-objects of the path; typically it points on the output *..DataOut* of the preceding module (e.g. the SMC_NCDecoder). It should be dimensioned appropriately (see above) !

nSizeOutQueue: UDINT

This variable contains the size of the data buffer, to which the list of GEOINFO structure objects will be written. This buffer must be at least five times as big as a GEOINFO structure, this means about 2KB. If this is not the case, SMC_NCDecoder will not execute any actions at all. The value can be predefined but later it only may be modified during a reset.

pbyBufferOutQueue: POINTER TO BYTE

This input must point to the first byte of the memory area which is allocated for the OUTQUEUE-structure. This area must be at least as big as defined in nSizeOutQueue. Typically the allocation of the memory buffer is done in the declaration part of the IEC-program by defining a byte-array (e.g. `BUF: ARRAY[1..10000] OF BYTE;` for a 10000 byte memory area). The value can be predefined but later may be modified only during a reset.

Outputs of the module:

This variable will be set to TRUE as soon as the input data of ...*DataIn* have been processed completely. Thereafter the module will not perform any further actions until it gets a reset. If input bExecute is FALSE, bDone will be reset to FALSE.

bError: BOOL

In case of an error this input gets TRUE.

wErrorID: SMC_ERROR (INT)

In case of an error, this input shows the error number.

poqDataOut: POINTER TO SMC_OUTQUEUE

This variable points to a SMC_OUTQUEUE-structure, which manages the SMC_GEOINFO-objects of the loopless path.

iStatus: AL_STATUS (INT)

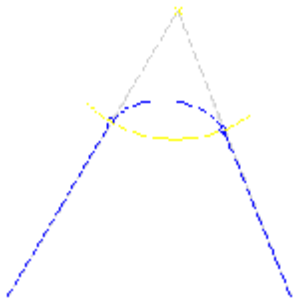
This enum-variable shows the current state of the module. Possible states:

AL_OFF	0	Avoid-Loop-functionality switched off
AL_ON	1	Avoid-Loop-functionality switched on
AL_END	2	Processing of the objects terminated

6.2.4 SMC_SmoothPath

The *SMC_SmoothPath* function block can be used for path-preprocessing. It smoothes **path angles**, thus creating a smooth path (slur path). This is needed for applications where the exactness of the path course is not as important as the velocity, and therefore angles, forcing to reduce the velocity to 0, must be avoided.

For this purpose the path will be cut in a defined distance to the angle and a **spline** will be inserted. The distance is given on the one hand by the SMC_GEOINFO-structure of the first object which should be smoothed, and on the other hand by one of its inputs. The sum of both values is taken as the radius of a circle which has its centre point in the path angle and which will intersect the surrounding objects.



SMC_SmoothPath is working as described in the following:

All SMC_GEOINFO-objects which are found in the Input-OUTQUEUE-structure will be checked one after the other. If in one of the objects Bit4 of the variable Intern_Mark is set, then the slurring will be started for all subsequent objects until in one of them Bit3 of Intern_Mark is set. The SMC_NCDecoder module will set these start and stop bits as a reaction to the instructions G51/G50. In other words: **The Smooth-Path functionality will be executed for all objects, which are placed between the instructions G51 and G50.**

Inputs of the function block:

bExecute: BOOL

The function block will do a reset and start the path correction (smoothing path) as soon as a rising edge is detected at this input.

bAppend: BOOL

As long as this input is FALSE, at each reset the DataOutQueue will be cleared. As long as it is TRUE, newly incoming data will be written to the end of the DataOutQueue.

poqDataIn: POINTER TO SMC_OUTQUEUE

This variable points to the SMC_OUTQUEUE-structure object, which contains the SMC_GEOINFO-objects of the unsmoothed path; typically it points to the output ..*DataOut* of the preceeding module (e.g. the SMC_NCDecoder).

bEnable: BOOL

If this input is set to FALSE (default), the module is in waiting status, this means that it will not perform any actions. With TRUE it will start resp. continue to slur the path.

bReset: BOOL

By setting this variable to TRUE the module will be reset to the start state. Thereby also the content of the SMC_OUTQUEUE-structure *DataOut* will be removed. Thereafter the module cannot get active again until *bReset* is reset to FALSE (default).

dEdgeDistance: LREAL

This input variable contains the value, which – added to the corresponding *ToolRadius*-value of the SMC_GEOINFO-object – determines the (minimum) distance to an angle, at which the particular objects will be cut and replaced by a spline (see above). This value can be modified online. Thus it is possible to predefine offline (by the SMC_GEOINFO-structure) and to modulate online. Default: 0.

dAngleTol: REAL

This input describes the angle tolerance value (see chapter 3.3) , up to which a path bend should not be smoothed.

nSizeOutQueue: UDINT

This variable contains the size of the data buffer, to which the list of GEOINFO structure objects will be written. This buffer must be at least five times as big as a GEOINFO structure, this means about 2KB. If this is not the case, SMC_NCDecoder will not execute any actions at all. The value can be predefined but later it only may be modified during a reset.

pbyBufferOutQueue: POINTER TO BYTE

This input must point to the first byte of the memory area which is allocated for the SMC_OUTQUEUE-structure. This area must be at least as big as defined in *nSizeOutQueue*. Typically the allocation of the memory buffer is done in the declaration part of the IEC-program by defining a byte-array (e.g. `BUF: ARRAY[1..10000] OF BYTE;` for a 10000 byte memory area. The value can be predefined but later it only may be modified during a reset.

Outputs of the function block:

bDone: BOOL

This variable will be set to TRUE as soon as the input data from ..*DataIn* are processed completely. Thereafter the module will not perform any further actions until it gets reset. If input bExecute is FALSE, bDone will be reset to FALSE.

bError: BOOL

In case of an error this input gets TRUE.

wErrorID: SMC_ERROR (INT)

In case of an error, this input shows the error number.

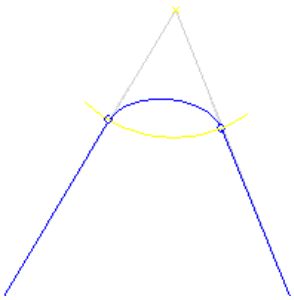
poqDataOut: POINTER TO SMC_OUTQUEUE

This output variable points to a SMC_OUTQUEUE-structure, which manages the slured SMC_GEOINFO-objects.

6.2.5 SMC_RoundPath

The *SMC_RoundPath* function block is very similar to the SMC_SmoothPath module. It rounds **angles**, which result at the junction of two lines, by **circular arcs**.

For this purpose the path will be cut in distance "r" to the angle and a **spline** will be inserted. The distance is given on the one hand by the SMC_GEOINFO-structure object of the first object which should be smoothed, and on the other hand by the input variable **dRadius.**



The value of dRadius is dominant. This means that only if dRadius=0 the value of the object will be regarded. If the defined value is higher than the half length of one of the both SMC_GEOINFO-objects, then the half length will be used.

The function block *SMC_RoundPath* is working as described in the following:

All SMC_GEOINFO-objects which are found in the Input-OUTQUEUE-Structure will be checked one after the other. If in one of the objects Bit5 of the variable Intern_Mark is set, then starting there angles will be rounded as long as in one of the subsequent objects Bit3 of Intern_Mark is set. The SMC_NCDecoder module will set these bits as a reaction to the instructions G52/G50. In other words: **The Round-Path functionality will be executed for all objects, which are placed between the instructions G50 and G51.**

Inputs of the module:

bExecute: BOOL

The function block will do a reset and start the path correction (rounding path) as soon as a rising edge is detected at this input.

bAppend: BOOL

As long as this input is FALSE, at each reset the DataOutQueue will be cleared. As long as it is TRUE, newly incoming data will be written to the end of the DataOutQueue.

poqDataIn: POINTER TO SMC_OUTQUEUE

This variable points to the SMC_OUTQUEUE-structure object, which contains the SMC_GEOINFO-objects of the unsmoothed path; typically on the output ..*DataOut* of the preceding module (e.g. the SMC_NCDecoder).

bEnable: BOOL

If this input is set to FALSE (default), the module currently is in waiting status, this means that it will not perform any actions. Getting TRUE it will start resp. continue to round the path.

bReset: BOOL

By setting this variable to TRUE the module will be reset to the start state. Thereby also the content of the SMC_OUTQUEUE-structure ..*DataOut* will be removed. Thereafter the module cannot get active again until bReset is reset to FALSE (default).

dRadius: LREAL

This input variable contains the value, which defines the (minimum) distance to an angle, at which the particular objects will be cut and replaced by a spline (see above). This value can be modified online. Thus it is possible to predefine offline (by the SMC_GEOINFO-structure) and to modulate online. Default: 0.

dAngleTol: REAL

This input describes the angle tolerance value, up to which a path bend should not be smoothed..

nSizeOutQueue: UDINT

This variable contains the size of the data buffer, to which the list of GEOINFO structure objects will be written. This buffer must be at least five times as big as a GEOINFO structure, this means about

2KB. If this is not the case, SMC_NCDecoder will not execute any actions at all. The value can be predefined but later it may only be modified during a reset.

pbyBufferOutQueue: POINTER TO BYTE

This input must point to the first byte of the memory area which is allocated for the SMC_OUTQUEUE-structure. This area must be at least as big as defined in *nSizeOutQueue*. Typically the allocation of the memory buffer is done in the declaration part of the IEC-program by defining a byte-array (e.g. `BUF: ARRAY[1..10000] OF BYTE;` for a 10000 byte memory area). The value can be predefined but later it may only be modified during a reset.

Outputs of the module:

bDone: BOOL

This variable will be set to TRUE as soon as the input data from ..*DataIn* are processed completely. Thereafter the module will not perform any further actions until it gets reset. If input bExecute is FALSE, bDone will be reset to FALSE.

bError: BOOL

In case of an error this input gets TRUE.

wErrorID: SMC_ERROR (INT)

In case of an error, this input shows the error number.

poqDataOut: POINTER TO SMC_OUTQUEUE

This output variable points on a SMC_OUTQUEUE-structure, which manages the rounded SMC_GEOINFO-objects.

6.2.6 SMC_CheckVelocities

This module checks the velocities of the particular path segments. If the OutQueue has not been created by the editor, but by the IEC program (e.g. by SMC_NCDecoder), this module must be called immediately before the Interpolator.

The main task of this function is to check the path for sharp bends and to reduce the velocity to zero at those.

Inputs of the module:

bExecute: BOOL

Bei steigender Flanke wird die Überprüfung begonnen.

poqDataIn: POINTER TO SMC_OUTQUEUE

This input points to the SMC_OUTQUEUE structure object, which describes the SMC_GEOINFO objects of the path; typically it points to the output poqDataOut of the preceeding module (e.g. SMC_NCDecoder/SMC_SmoothPath).

dAngleTol: REAL

This input describes the tolerance angle (angle leeway) up to which at a sharp bend of the path no stop should be done.

Outputs of the module:

bError: BOOL

Gets TRUE in case of an error.

wErrorID: SMC_ERROR (INT)

In case of an error, this output shows the error number.

poqDataOut: POINTER TO SMC_OUTQUEUE

This output points on the SMC_OUTQUEUE structure object, which contains the path with the permissible velocity values and now should be fed to the Interpolator.

SMC_LimitCircularVelocities

This module (SM_CNC.lib) checks the particular elements of the OutQueue and limits the path velocities of circular elements against their radii. The path acceleration - if moving with constant velocity (v) across an transition from a line to a circle with radius r - according to amount will jump from 0 to value $\{ EQ \} \{ EQ \r(v^2;r) \}$. In order to limit this acceleration jump to value Acc, the velocity of the arc at the transition $v = \{ EQ \r(a * r) \}$ must not exceed.

The module controls the transition of two elements (line on circular arc, circular arc on line and circular arc on circular arc) and adapts the end velocity of the first element so that the acceleration jump does not exceed the value **dMaxAccJump**.

Further on the module limits the path acceleration on circles on value **dMaxAcc** by appropriately reducing the path-velocity of the circle.

Inputs of the module:

bExecute: BOOL

The module will perform a reset and will start tool radius correction, as soon as this input gets a rising edge.

bAppend: BOOL

If FALSE, at each reset the output data queue DataOut will be cleared. If TRUE, the new data will be written to the end of the DataOut queue.

poqDataIn: POINTER TO SMC_OUTQUEUE

This input points on the SMC_OUTQUEUE structure object, which contains the SMC_GEOINFO objects of the path to be modified; typically it points on output DataOut of the preceding module (e.g.SMC_NCDecoder).

dMaxAcc: LREAL

This input variable gives the maximum acceleration value permissible for circular arcs. A value equal 0 will cause that no check will be done.

dMaxAccJump: LREAL

This input variable gives the maximum acceleration jump (a) for a transition of two objects. A value equal 0 will cause that no check will be done.

dVmaxPerRadiusUnit: LREAL

This input variable contains the value Acc4π² (see above), i.e. the maximum velocity of the arc at a radius of 1.

nSizeOutQueue: UDINT

This variable tells about the size of the data buffer, to which the list of rounded GEOINFO structure objects is written. This must be five times as big as a SMC_GEOINFO structure, thus must have a size of ca. 2KB. If this is not the case, the SMC_SmoothPath module will not perform any actions. The value can be set but only may be modified during a reset afterwards.

pbyBufferOutQueue: POINTER TO BYTE

This input must point on the first byte of the memory available for the OUTQUEUE structure. This memory area at least must be as big as defined in Size_SMC_OUTQUEUE. Typically the the storage allocation is done in the declaration part of the IEC program via a byte-array (e.g. BUF: ARRAY[1..10000] OF BYTE; for a memory of 10000 Byte). Also this value can be set but only may be modified afterwards during a reset.

Outputs of the module:

bDone: BOOL

This variable is set TRUE as soon as the input data from DataIn have been processed completely. Afterwards the module will not perform any further action until a reset. If bExecute-Eingang gets FALSE, bDone will be reset to FALSE.

bError: BOOL

In case of an error this variable gets TRUE.

wErrorID: SMC_ERROR (INT)

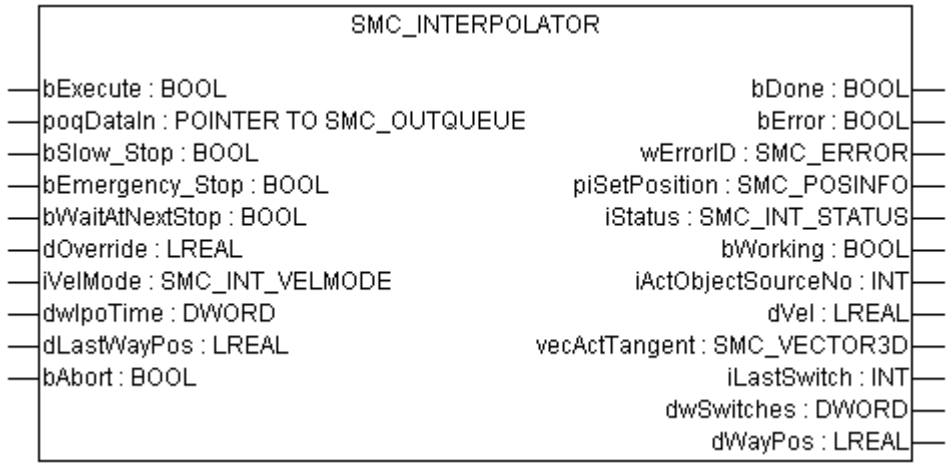
In case of an error this variable gets the error number.

poqDataOut: POINTER TO SMC_OUTQUEUE

This output points on a SMC_OUTQUEUE structure managing the new SMC_GEOINFO objects.

6.2.7 SMC_Interpolator

The SMC_Interpolator function block is used to convert a continuous path which is described by SMC_GEOINFO-objects, into **discrete path position points**, thereby regarding a defined velocity profile and time pattern. These position points typically afterwards will be transformed by the IEC-program (e.g. to drive-axis-positions) and sent to the drives.



Inputs of the function block:

bExecute: BOOL

The function block will do a reset and start the path correction (rounding path) as soon as a rising edge is detected at this input.

poqDataIn: POINTER TO SMC_OUTQUEUE

This variable points to the SMC_OUTQUEUE-structure object, which contains the SMC_GEOINFO-objects of the unsmoothed path; typically it points to the output ..*DataOut* of the preceeding module SMC_CheckVelocities.

bSlow_Stop: BOOL

If this variable is set to FALSE (default), the path will be passed non-stop. With TRUE the SMC_Interpolator will be caused to reduce the velocity to 0 – according to the defined velocity profile (*byVelMode*, see below) and the maximum delay of the current GEOINFO-object (d*Decel*, see below) – and to wait until bSlow_Stop will be reset to FALSE.

bEmergency_Stop: BOOL

This input per default is FALSE. As soon as it gets TRUE, the SMC_Interpolator will cause an immediate stop, this means that the position will be retained. Thus the velocity will be set to 0 immediately.

bWaitAtNextStop: BOOL

As long as this variable is FALSE (default), the path is passed non-stop. With TRUE the SMC_Interpolator will be caused to retain the position at the next regular stop – this means at position

points where the velocity is 0, typically at path angles – and to pause until *bWaitAtNextStop* will be reset to FALSE.

dOverride: LREAL

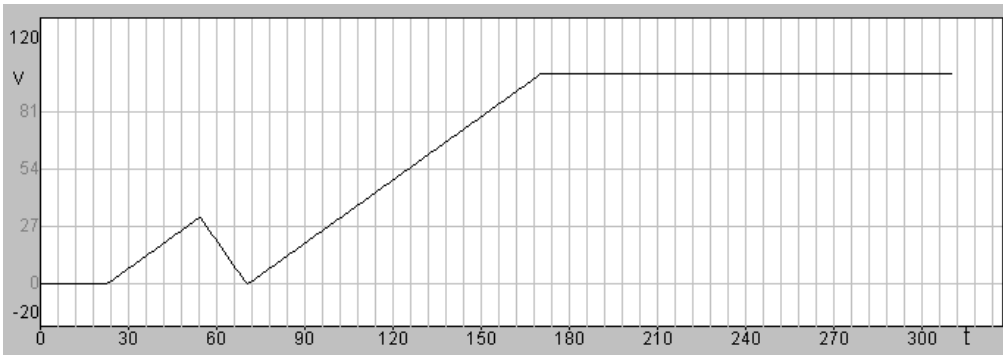
This variable can be used to handle the override. Valid values are higher than 0.01. *dOverride* is multiplied with the scheduled velocity of the particular objects and thus allows to increase resp. reduce the scheduled velocity in online mode. For example dOverride=1 (default) effects that the programmed scheduled velocities will be executed, while an dOverride=2 would double them.

Please regard: The override can be modified at any time, but the modification will only be applied, if currently no acceleration or deceleration is in progress !

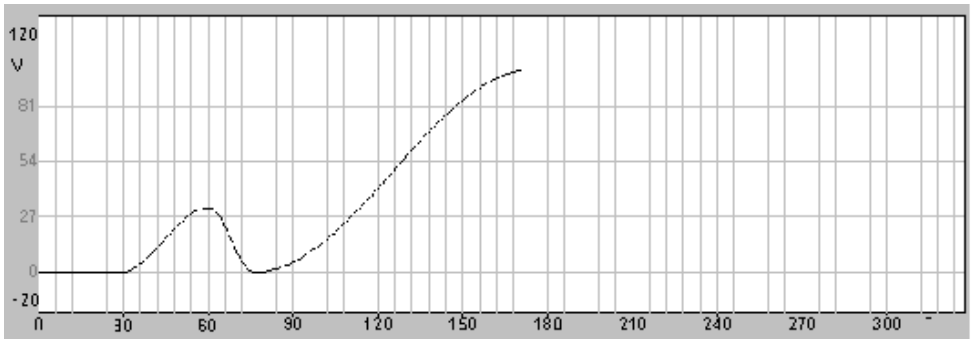
iVelMode: SMC_INT_VELMODE

This input defines the velocity profile. The value "TRAPEZOID" (default) effects a velocity profile which has a trapezoid form, "SIGMOID" one which has a S-form:

Example of a trapezoid velocity profile (byVel_Mode = 0):



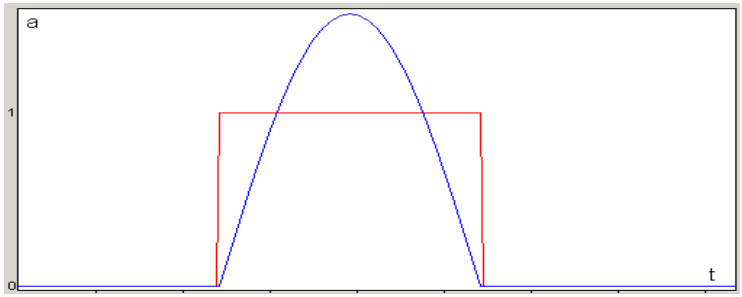
Example of a sigmoid velocity profile (byVel_Mode = 1):



In the examples shown above the maximum acceleration (*Accel*) is lower than the maximum deceleration (*Decel*). This causes the different slope values of the velocity curve at acceleration and deceleration.

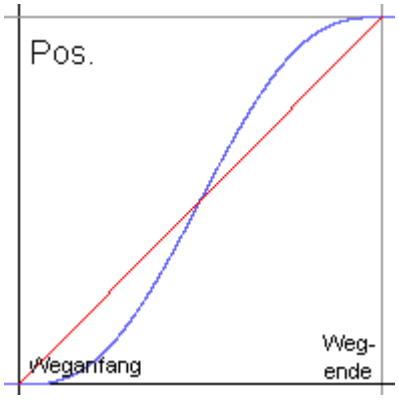
The advantage of a sigmoid velocity profile is that the associated acceleration – in contrast to the trapezoid – is continuous and thereby brings a relief especially for heavy machines. This must be paid by a slightly increased calculating time.

As the sigmoid velocity profile (blue) is designed in a way that changing to the trapezoid profile (red) does not result in a change of the time needed for the passing the complete path, the restrained increase of the acceleration at the beginning and the end must be compensated by a higher acceleration in the midway. Thereby you must regard, that the maximum acceleration resp. deceleration, which is programmed in the SMC_GEOINFO-objects, will be exceeded in the maximum by the factor π/2:



Any online-change will not be applied – like described for *dOverride* – until a currently running acceleration or deceleration has been terminated.

In order to interpolate also the additional axes in the sigmoid form (blue, see drawing below) instead of linearly (red), the corresponding bits must be set in the variable *wSProfile of piStartPos* of the current object. This will effect that the additional axis does not get interpolated linearly concerning the path length in the X-,Y-,Z-space, but in a polynomial dependency on this path length, thus resulting in a sigmoid profile for the axis position, which has a velocity and acceleration of 0 at the beginning and at the end of a path segment.



dwIpoTime: DWORD

This variable, which must be set for **each call**, contains the cycle time in µsec.

dLastWayPos: LREAL

This input allows the user to measure the stretch of the path which is racked out by the Interpolator. Output dWayPos is the sum of dLastWayPos and the distance covered within the current cycle. If dLastWayPos=0, dWayPos shows the length of the current path segment. If dLastWayPos is set equal to output dWayPos, dWayPos always will be incremented by the current path segment and you get the total length of the already covered path. In doing so dLastWayPos at any time can be (re)set to 0 or a different value.

bSingleStep: BOOL

This input effects that the interpolator will stop at the end of each path object (also at transitions with identic tangent) and wait until this input is reset to FALSE. If bSingleStep is set TRUE during the move, the interpolator will stop at the end of that object, which it can reach without exceeding the scheduled deceleration value.

bAbort: BOOL

This input aborts the processing of a outline.

Outputs of the function block:

bDone: BOOL

This variable will be set to TRUE as soon as the input data (poqDataIn) have been processed completely. Hereafter the function block will not perform any further actions until a reset will be done. If input bExecute is FALSE, bDone will be reset to FALSE.

bError: BOOL

In case of an error this input gets TRUE.

wErrorID: SMC_ERROR (INT)

This enumeration variable may describe an error which has occurred during the interpolation run. After an error has occurred, the processing gets stopped until a Reset will be done.

Per each call the SMC_Interpolator – considering the predefined parameters, the velocity history and the last position – will calculate and put out the next point. As soon as the processing of currently first GEOINFO-object has been finished, it will be removed from the poqDataIn-SMC_OUTQUEUE-structure.

piSetPosition: SMC_POSINFO

This variable contains the target position which has been calculated according to the predefines. *Set_Position* is a SMC_POSINFO-structure and not only contains the Cartesian coordinates of the target point on the path, but also the position of the additional axes.

iStatus: INT_STATUS (INT)

This enumeration variable shows the current status of the function block. Possible states:

IPO_UNKNOWN	0	Internal state. This state may not occur after a complete pass of the SMC_Interpolator.
IPO_INIT	1	Module is in initialization state; DataIn currently is not full and also not yet has been full.
IPO_ACCEL	2	Module currently is accelerating.
IPO_CONSTANT	3	Module currently is moving with constant velocity.
IPO_DECEL	4	Module currently is decelerating.
IPO_FINISHED	5	Processing of the GEOINFO list is terminated. Any further GEOINFO objects which arrive subsequently in *DataIn* will not be processed.
IPO_WAIT	6	Module is waiting, because one of the following situations has occurred: *Emergency_Stop* = TRUE *Slow_Stop* = TRUE and *Vel* = 0 *Wait_At_Next_Stop* = TRUE and *Vel* = 0

bWorking: BOOL

This output gets TRUE, as soon as the processing of the list has been started but is not yet finished (IPO_ACCEL or IPO_CONSTANT or IPO_DECEL or IPO_WAIT). Otherwise *bWorking* is FALSE.

iActObjectSourceNo: INT

Here you find the value of *SourceLine_Nr* of the currently passed GEOINFO-object of the *DataIn*-queue. If the SMC_Interpolator does not work (any longer) (*Working* = FALSE), the value is "–1".

dVel: LREAL

This variable contains the current velocity which results if an object is moving from the preceeding position to *Set_Position* within the given time *Ipo_Time*.

vecActTangent: SMC_VECTOR3D

This structure contains the direction of the path valid for the position *Set_Position*. If *Vel* = 0 , *vecAct_Tangent* is filled up with zeros.

iLastSwitch: INT

This output shows the number of the last passed switch. Regard: If several switches have been passed within one cycle, only the last one will be shown.

dwSwitches: DWORD

This DWORD describes the current switch status of all switches 1 – 32. Bit0 of the DWORD represents switch1, Bit31 represents switch321, Bit31 for auxiliary mark 32. Thereby, in contrast to *iLastSwitc*h, it can be avoided that any switch is not regarded.

dWayPos: LREAL

For a description see above: input dLastWAyPos.

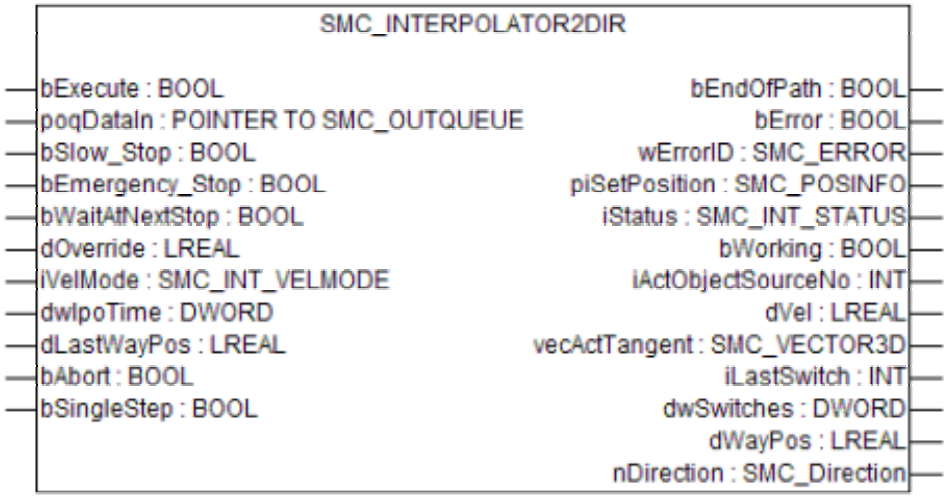
At each call the SMC_Interpolator will calculate and provide - regarding the given parameters, the velocity history and the last path position - the subsequent path position point. If the first GEOINFO object has been processed, it will be removed from the poqDataIn-SMC_OUTQUEUE structure.

Please regard: At the end of a path run the variable SMC_OUTQUEUE is empty. If you want to process the same outline once more, either you have to transform the CNC program via Decoder and Path-Preprocessing-Modules to a SMC_OUTQUEUE structure, or you have to use the function SMC_RESTOREQUEUE (also part of SM_CNC.lib). The latter is only possible, if the OUTQUEUE buffer is so big that it can catch the complete path.

6.2.8 SMC_Interpolator2Dir

This module corresponds concerning function and allocation of its inputs and outputs to FB SMC_Interpolator, with the difference that it can also reversely interpolate a path.

For this purpose input **dOverride** gets assigned a negative value, which makes SMC_Interpolator2Dir interpolating in negative direction. For example this input can get assigned the analog velocity input of a hand wheel, allowing the user to move forward and backward with a desired velocity.



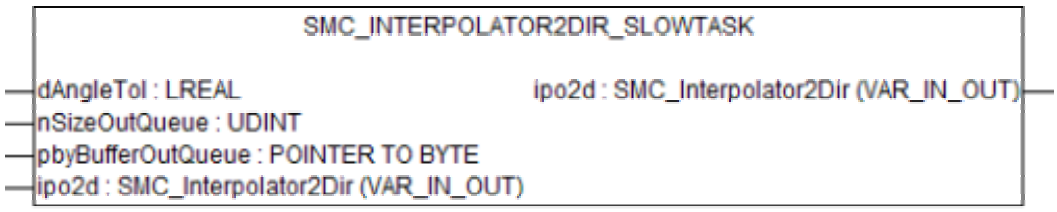
Additional inputs and outputs of the module:

nDirection: SMC_Direction

The module outputs in which direction it is currently. Possible values: IPO_positive, PO_negative and IPO_standstill.

There are the following preconditions for the use of the module:

1. The path must be completely go in poqDataIn. Due to the fact that the module must alternately move forward and backward, the complete path must be available in the memory.
2. An additional module, **SMC_Interpolator2Dir_SlowTask**, gets called:



This module is responsible for generating the backward-path. It has been split off from SMC_Interpolator2Dir and thus can be out housed to a lower-prioritized task on fully stretched systems with low performance.

Inputs of the module:

dAngleTol: LREAL

Angle tolerance fort he backward path. Typically identic to the angle tolerance of the original path.

nSizeOutQueue: UDINT, pbyBufferOutQueue: POINTER TO BYTE

Size and pointer on data buffer, to which the backward path should be stored. Must be at least that big that the complete path goes in.

Ipo2d: SMC_Interpolator2Dir

SMC_Interpolator2Dir-instance, for which the backward path should be generated.

6.3 Auxiliary Modules for Path Rotations, Translations and Scalings

The function blocks SMC_RotateQueue2D and SMC_TranslateQueue3D rotate resp. translate the path which is stored in the SMC_OUTQUEUE.

The input variable ***poqDataIn*** is the pointer to the structure SMC_OUTQUEUE which describes the path to get rotated resp. translated.

The input variable ***bEnable***, initialized with FALSE, will avoid the rotation resp. translation of the path until it is set to TRUE. Then all GEOINFO-objects found in *poqDataIn* will be processed. As soon as *bEnable* gets FALSE, the modules will not execute any further modifications.

The input variable ***bReset***, also initialized with FALSE, effects that the GEOINFO-objects which are currently found in the *poqDataIn*, will not get rotated resp. translated, but only those which additionally come in as from now.

- **SMC_ROTATEQUEUE2D**
  The path stored in poqDataIn will be rotated around the Z-axis by the angle given by dPhi [°]. A positive angle effects a positive rotation in mathematical sense (counterclockwise).

- **SMC_TRANSLATEQUEUE3D**
  The path stored in poqDataIn will be translated according to the vector given by vec, which is of structure type SMC_VECTOR3D (see SMC_VECTOR3D).

- **SMC_SCALEQUEUE3D**
  The path contained in poqDataIn will be stretched by factor fScaleFaktor.

As a sudden modification of the characteristic parameters of the rotation/translation (*dPhi*, *vec*) during the processing might cause a break in the path, any changes of the corresponding inputs will not be regarded until the SMC_OUTQUEUE is empty at *poqDataIn*, or a Reset (*bReset*) will be done.

In order to effect a rotation in the (XY)-plane around another point than (00), this means in order to reach the point (XpYp), use a sequence of a translation per the vector (-Xp-Yp°0) , the rotation per the desired angle ***dPhi*** and a further translation per the vector (XpYp°0) .

6.4 Settings via global variables

In „SoftMotion_CNC_Globals“there are defined several internal variables and constants. Some of them can be modified:

Examination for zero (see 3.6, ‚Set epsilon values…'):

- g_fSMC_CNC_EPS (Epsilon value for accurate examination for zero)
- g_fSMC_CNC_EPS_RELUCTANT (Epsilon value for tolerant examination for zero)

6.5 Structures in the SM_CNC.lib

See in the following a selection of structures provided by the library SM_CNC.lib, which are used by the library modules for managing the position data, the path segments (objects) and vectors: SMC_POSINFO, SMC_GEOINFO, SMC_VECTOR3D, SMC_VECTOR6D. Further on the SMC_OUTQUEUE-structure allows to manage GEOINFO-objects in a list of defined size.

SMC_POSINFO

This structure, which is part of the SMC_CNC.lib, describes the coordinates and the position of the additional axes for the particular position points.

```
TYPE SMC_POSINFO:
STRUCT
       iFrame_Nr:INT;
       wAuxData:WORD;
       wSProfile:WORD;
       dX:LREAL;
       dY:LREAL;
       dZ:LREAL;
       dA:LREAL;
       dB:LREAL;
       dC:LREAL;
       dA1:LREAL;
       dA2:LREAL;
       dA3:LREAL;
       dA4:LREAL;
       dA5:LREAL;
       dA6:LREAL;
END_STRUCT
END_TYPE
```

The variables **dX**, **dY** and **dZ** describe the position in the coordinate system, **dA1**, ..., **dA6** describe the position of the additional axes. In **iFrame_Nr** the user can store further information, which is not relevant for the SoftMotion modules. **dA**, **dB** and **dC** currently are not used.

wAuxData describes bit by bit which of the position axes should be calculated by the SMC_Interpolator. **wAuxData** will be initialized with 2#111, that means that X-, Y- and Z-axis get interpolated. If the first bit is set, the dX-position gets calculated, Bit 7 for example effects a processing of dA2.

wSProfile in the same way describes for the additional axes (all besides X,Y-axes), whether they should be interpolated by the Interpolator module linearly (FALSE) or in sigmoid (S-) shape (TRUE). Bit2 stands for the Z-axis, Bit6 for P, Bit7 for Q, Bit8 for U, Bit9 for V and Bit10 for W.

SMC_GEOINFO

This structure, which is part of the SMC_CNC.lib, contains the path objects. A (path) object is a segment of the programmed path, which due to its geometrical properties completely can be stored in the following structure:

```
TYPE SMC_GEOINFO:
STRUCT
       iObjNo:INT;
       iSourceLineNo:INT;
       iMoveType:MOVTYP;
```

```
        piStartPos:SMC_POSINFO;
        piDestPos:SMC_POSINFO;
        dP1:LREAL;
        dP2:LREAL;
        dP3:LREAL;
        dP4:LREAL;
        dP5:LREAL;
        dP6:LREAL;
        dP7:LREAL;
        dP8:LREAL;
        dT1:LREAL;
        dT2:LREAL;
        dToolRadius:LREAL;
        dVel:LREAL;
        dVelEnd:LREAL;
        dAccel:LREAL;
        dDecel:LREAL;
        dLength:LREAL;
        byInternMark:BYTE;
        dHelpPos: ARRAY[0..MAX_IPOSWITCHES] OF LREAL;
        iHelpID: ARRAY[0..MAX_IPOSWITCHES] OF INT;
END_STRUCT
END_TYPE
```

iObjNo: INT

This integer value describes any desired object number. It has no meaning for the actual path description.

iSourceLineNo: INT

This integer value typically describes the source code line number of the CNC program. It has no meaning for the actual path description.

iMoveType: MOVTYP (INT)

This enumeration contains the following valid values and describes the object type:

LIN	1	straight movement (G01)
CLW	2	circle in clockwise direction (G02)
CCLW	3	circle in counterclockwise direction (G03)
SPLINE	5	spline, parable (G05, G06)
ELLCLW	8	Ellipse clockwise (G08)
ELLCCLW	9	Ellipse counterclockwise (G09)
LINPOS	100	straight positioning (G00)
INITPOS	110	blind positioning (start point not yet known; a continuous position will be added by the SMC_Interpolator)

piStartPos: SMC_POSINFO

This structure describes the exact start position of the object. (will be ignored if *Move_Type* = INITPOS).

piDestPos: SMC_POSINFO

This structure describes the exact end position of the object.

dP1, ..., dP8: LREAL

These variables contain, depending on the Move_Type (see above), further path describing information:

LIN LINPOS	not relevant, because the complete information already is contained in *Start_Pos* and *Dest_Pos*
CLW	dP1: X-coordinate of the circle centre

CCLW	dP2: Y-coordinate of the circle centre dP3: Circle radius
SPLINE	spline parameter
ELLCLW, ELLCCLW	P1: X-Coordinate of the circle centre P2: Y-Coordinate of the circle centre P3: X-Component of the mainaxis-1-vector P4: Y-Component of the mainaxis-1-vector P5: Length of the main axis P6: Length of the sub-axis P7: Direction of the main axis (°) P8: Ratio P6/P5
INITPOS	not relevant

dT1, dT2: LREAL

These variables contain the start and the end position of the parameter. Depending on the Move_Type this means:

LIN LINPOS	not relevant, because the complete information already is contained Start_Pos and Dest_Pos
CLW	dT1: start angle in mathematical sense (degree): (0 = East, 90 = North, 180 = West, 360 = South) dT2: apex angle of the circle, length of the circular arc (degree): (e.g.: 90=quarter of a circle, 180 =semicircle)
CCLW	dT1: start angle in mathematical sense (degree): (0 = East, 90 = North, 180 = West, 360 = South) dT2: negative apex angle of the circle: (e.g.: -90=quarter of circle, -180 = semicircle)
SPLINE	start- and end value of parameter "t" (see description of a spline). Default: 0 and 1.
INITPOS	not relevant

dToolRadius: LREAL

This variable contains the information necessary for the path-preprocessing (see The SMC_ToolCorr Module, The SMC_RoundPath Module). The entry is without any meaning if none of the appropriate path-preprocessing modules is called (SMC_ToolCorr, SMC_RoundPath).

dVel, dVelEnd: LREAL

These variables, which must be assigned in any case, contain information on the velocity profile of the object. *dVel* describes the target velocity, which should be reached, *dVelEnd* describes the velocity which must be run at the end of the object (see The SMC_Interpolator Module) (path units/sec).

dAccel, dDecel: LREAL

dAccel describes the maximum allowed acceleration, dDecel the maximum allowed deceleration (path units/sec^2). Both variables are predefined with "100".

dLength: LREAL

This variable, which must be assigned in any case, contains the length of an object (path units).

byIntern_Mark: BYTE

This variable describes start and end of path-preprocessing actions:

Bit 0 set	Stop tool radius correction after this object
Bit 1 set	Start tool radius correction to the left of this object
Bit 2 set	Start tool radius correction to the right of this object
Bit 3 set	Stop path rounding/smoothing after this object
Bit 4 set	Start path rounding/smoothing at this object
Bit 4 set	Start path rounding at this object
Bit 6 set	Stop avoiding loops after this object
Bit 7 set	Start avoiding loops at this object

dHelpPos: ARRAY[0..MAX_IPOSWITCHES] OF LREAL,

iHelpID: ARRAY[0..MAX_IPOSWITCHES] OF INT:

These variables describe the relative position (0: Object start, 1:Object end; similar to G-Code: O) and the ID (cp. G-Code: H) of the auxiliary switches.

SMC_VECTOR3D

This structure, which is part of the SM_CNC.lib, describes a three-dimensional vector:

```
TYPE SMC_VECTOR3DQ
STRUCT
       dXQLREALR
       dYQLREALR
       dZQLREALR
END_STRUCT
END_TYPE
```

SMC_VECTOR6D

This structure, which is part of the SM_CNC.lib, describes a six-dimensional vector:

```
TYPE SMC_VECTOR6DQ
STRUCT
       dXQLREALR
       dYQLREALR
       dZQLREALR
       dAQLREALR
       dBQLREALR
       dCQLREALR
END_STRUCT
END_TYPE
```

SMC_OUTQUEUE and its Functions

This structure, which is part of the SM_CNC.lib, can be used to manage GEOINFO-objects in a list of defined size.

```
TYPE SMC_OUTQUEUE Q
STRUCT
       wOUTQUEUEStructIDQ WORDR
       pbyBufferQ POINTER TO BYTER
       nSizeQ UDINTR
       nReadPosQ UDINTR
       nWritePosQ UDINTR
       bFullQ BOOLR
       bEndOfListQ BOOLR
       byGeneratorQ BYTER
```

```
END_STRUCT
END_TYPE
```

Via variable *wOUTQUEUEStructID*, which has a fix value, the modules internally check whether the provided structure variable is of type SMC_OutQueue.
The variable byGenerator describes the originator of the queue. This information is used by the Interpolator to check whether module SMC_CheckVelocities has been processed as last one as prescribed. The following values are defined:

originator	value
SMC_NCDecoder	1
SMC_AvoidLoop	10
SMC_LimitCircularVelocity	11
SMC_RoundPath	12
SMC_SmoothPath	13
SMC_ToolCorr	14
SMC_RotateQueue2D	30
SMC_ScaleQueue3D	31
SMC_TranslateQueue3D	32
SMC_CheckVelocities	254
CNC-Editor	255

The SoftMotion-library SM_CNC.lib provides the following modules for the handling of a SMC_OUTQUEUE-structure object:

BOOL **SMC_RESTOREQUEUE**(Enable: BOOL, POQ: POINTER TO SMC_OUTQUEUE)

This function restores an already interpolated resp. otherwise processed structure. This is only possible, if the list at POQ can contain the complete path.

POINTER TO SMC_OUTQUEUE SMC_APPENDOBJ(POQ: POINTER TO SMC_OUTQUEUE, PGI: POINTER TO SMC_GEOINFO)

This boolean function appends the GEOINFO object, which is passed by *PGI*, to the end of the list (*POQ*), if this list has been initialized correctly and is not yet filled completely. In case of success the function returns a pointer to the new list element, otherwise 0.

BOOL SMC_DELETEOBJ(POQ: POINTER TO SMC_OUTQUEUE, N: UINT)

This boolean function deletes the N-th object from the list (*POQ*), whereby counting starts with 0. If N-1 is greater than the number of GEOINFO objects stored in the list, nothing will happen and FALSE will be returned, otherwise TRUE.

UINT SMC_GETCOUNT(POQ: POINTER TO SMC_OUTQUEUE)

This function of data type UINT returns the number of objects which are stored in the SMC_OUTQUEUE list (*POQ*).

POINTER TO SMC_GEOINFO GETOBJFROMEND(POQ: POINTER TO SMC_OUTQUEUE, N: UINT)

This function returns – provided that *POQ* is initialized correctly and that there are at least *N+1* elements – a pointer to the *N*-th *GEOINFO* object (start counting from the end) of the list (*POQ*); so for *N=0* the last list element would be returned.

Initialization of the structure:

The SoftMotion modules *SMC_NCDecoder*, *SMC_SmoothPath*, *SMC_RoundPath*, *SMC_AvoidLoop* and *SMC_ToolCorr*, which provide a pointer to an internally handled OUTQUEUE-structure, automatically will do the initialization of this structure. The modules *SMC_SmoothPath*, *SMC_RoundPath*, *SMC_ToolCorr*, *SMC_AvoidLoop* and *SMC_Interpolator* need the pointer on a correct OUTQUEUE list as an input. If this list is programmed and filled "manually", then the correct initialization also must be done manually. For this purpose the first two variables (buffer, size) have to be set. **It is strictly recommended** to use the above described functions for working with a

SMC_OUTQUEUE structure, and to avoid – after the initialization – any modifications of the other parameters.

Components of the structure:

pbyBuffer: POINTER TO BYTE

This variable contains the address of a coherent memory buffer, which is allocated for the GEOINFO-objects. This buffer must be assigned in the IEC program and its address then be written to this variable. The assignment in the declaration part e.g. can be done by using a byte-array (BUF: ARRAY[1..10000] OF BYTE; for a 10000 byte memory area).

nSize: UDINT

The size of the memory area which is allocated by *pbyBuffer*.

nReadPos: UDINT

Relative address of the first object in the list (referring to the first byte of the allocated memory buffer).

nWritePos: UDINT

Relative Address of the first free byte following the object list (referring to the first byte of the allocated memory buffer).

bFull: BOOL

This variable will be set to TRUE by the function APPENDOBJ as soon as the list is filled up and only space is left for three further GEOINFO objects (safety buffer). DELETEOBJ will set it back to FALSE as soon as elements get removed from the list.

bEndOfList: BOOL

The SoftMotion modules, which get an OUTQUEUE-structure as an input, do not start the processing of this queue until it has been filled completely in order to avoid e.g. a data underrun (SMC_Interpolator). Due to the fact that during working on the last SMC_GEOINFO-objects of a path the OUTQUEUE is not full any longer, *bEndOfList* must be set to TRUE as soon as the last SMC_GEOINFO-object has been stored, in order to keep the processing running. If the list is empty thereafter, but should be filled up again, *bEndOfList* must be reset to FALSE.

SMC_CNC_REF

In this data structure parsed G-Code-files are managed:

```
TYPE SMC_CNC_REF :
STRUCT
       wCNCREFStructID: WORD := 16#BA56;
       nElements: UDINT;
       diReadPos: UDINT := 0;
       udiBuffer: UDINT := 16#FFFFFFFF;
       pgc: POINTER TO SMC_GCODE_WORD := 0;
       piStartPosition: SMC_POSINFO;
       strProgramName:STRING := '';
END_STRUCT
END_TYPE
```

Via variable *wCNCREFStructID*, which has a fix values, the modules check internally, whether the consigned structure variable is of type SMC_CNC_REF.

Variable *pgc* points on the first SMC_GCODE_WORD.

nElements contains the number of SMC_GCODE_WORD structures at *pgc*.

The start position of the CNC program is stored in *piStartPosition*, its name is stored in *strProgramName*.

The variable *diReadPos* and *udiBuffer* are used internally.

SMC_GCODE_WORD

In this data structure G-Code words are stored:

```
TYPE SMC_GCODE_WORD :
STRUCT
       byLetter:BYTE:=0;
       fValue:LREAL:=0;
       diValue:DINT:=0;
       pAdr:POINTER TO BYTE:=0;
       byVarType:BYTE:=0;
END_STRUCT
END_TYPE
```

byLetter gets the ASCII-code of the letter of the word, *fValue* and *diValue* get the value of this letter as floating point- and integer number. If instead of a fix values a variable is used, *pADR* contains a pointer on this variable and *byVarType* contains the variables' data type:

1	INT
2	BYTE
3	WORD
4	DINT
5	DWORD
6	REAL
14	SINT
15	USINT
16	UINT
17	UDINT
22	LREAL

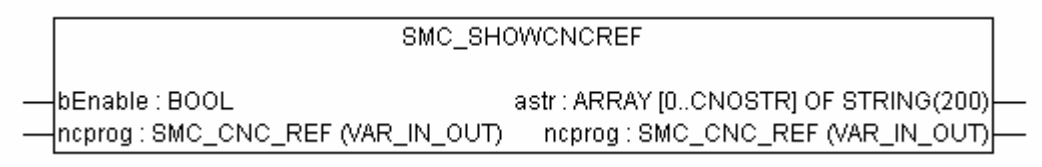
7 The library SM_CNCDiagnostic.lib

This library provides auxiliary modules which can be very useful during the implementation phase, because they help to display the data which are exchanged between the modules.

7.1 Function blocks for the analysis of SMC_CNC_REF data

7.1.1 The function block SMC_ShowCNCREF

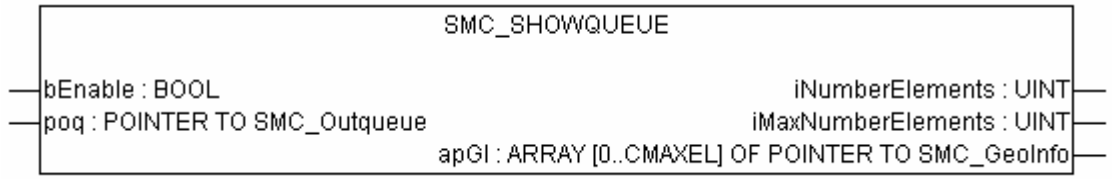
This module can display the first ten lines of a NC program, which is available in form of a data structure SMC_CNC_REF, in text strings (Din66025). The output is given in an array of string (*cnostr*) containing the text lines. The visualization template VISU_SMC_ShowCNCRef can display these outputs.



7.2 Function blocks for analysis of SMC_OutQueue data

7.2.1 The function block SMC_ShowQueue

This module provides the first ten SMC_GeoInfo objects of an OutQueue in form of an ARRAY OF POINTER TO SMC_GeoInfo. Some important elements can then be displayed by the visualization template VISU_SMC_ShowQueue. Among them are: object number, line number, object type, start position (X/Y/Z), end position (X/Y/Z), set velocity and final velocity.



8 The Library SM_Trafo.lib

8.1 Overview

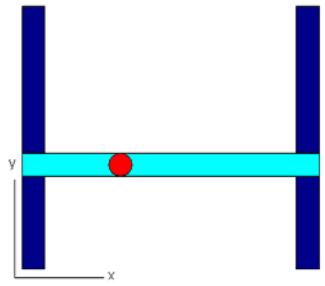
This library is an extension for the SM_CNC.lib and provides modules, which can be used for transformation of GEO- to drive coordinates and the axis control. There are modules which control drives with target values, and simultaneously watch the target values and detect jumps.

Besides that there are modules for mathematic forward and backward transformation for some usual kinematics. Instances of the forward-transformation-modules can be connected with the also provided visualization-templates, which allow an immediate and simple visualization.

8.2 Transformation function blocks

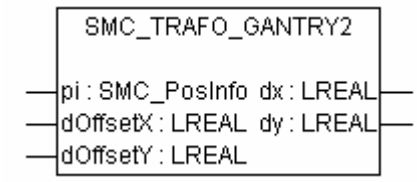
The modules which refer to a special kinematics belong together in pairs, whereby that module which is named SMC_TRAFO_<Kinematics>, proceeds a backward calculation, and that which is named SMC_TRAFOF_<Kinematics> proceeds a forward calculation. Each instance of a SMC_TRAFOF_<Kinematics>-module can be connected to a visualization template which is named SMC_VISU_<Kinematics>.

8.2.1 Portal Systems



For portal systems no transformation must be done, thus the modules just add offset on the x-, y- and z-axes.

SMC_TRAFO_Gantry2



pi: SMC_PosInfo

Target position vector. Output of the Interpolator.

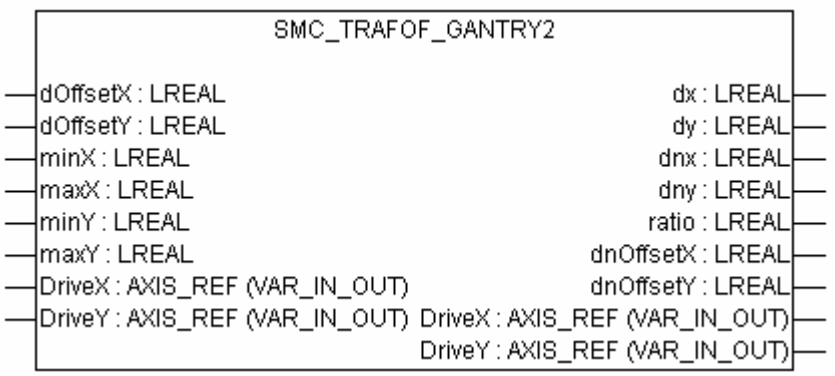
dOffsetX, dOffsetY: LREAL

Offset for x- and y-axis.

dx, dy: LREAL

Target values for x- and y-axis.

SMC_TRAFOF_Gantry2



dOffsetX, dOffsetY: LREAL

Offset for x- and y-axis. Same values as described for SMC_TRAFO_Gantry2.

minX, maxX, minY, maxY: LREAL

Move range (for visualization).

DriveX, DriveY: AXIS_REF

x-, y-axis.

dx, dy: LREAL

x-, y-position in Geo-coordinates.

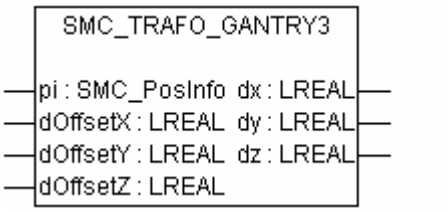
dnx, dny, dnOffsetX, dnOffsetY: LREAL

Standardized x- and y-position [0..1] and offsets (for visualization).

ratio: LREAL

Ratio of x-interval and y-interval (for visualization).

SMC_TRAFO_Gantry3



pi: SMC_PosInfo

Target position vector. Output of the Interpolator.

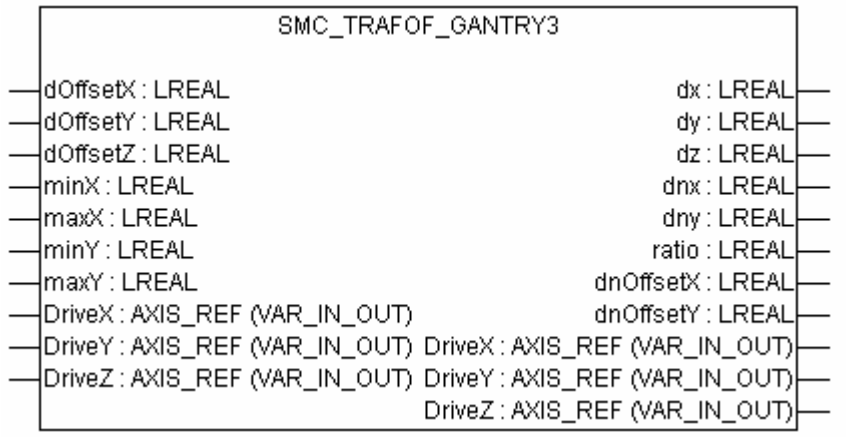
dOffsetX, dOffsetY, dOffsetZ: LREAL

Offset for x-, y- und z-axis.

dx, dy, dz: LREAL

Target values for x-, y- and z-axis.

SMC_TRAFOF_Gantry3



dOffsetX, dOffsetY, dOffsetZ: LREAL

Offset for x-, y- and z-axis. Same values as described for SMC_TRAFO_Gantry3.

minX, maxX, minY, maxY: LREAL

Move range (for visualization).

DriveX, DriveY, DriveZ: AXIS_REF

x-, y-, z-axis.

dx, dy, dz: LREAL

x-, y-, z-position in GEO-coordinates.

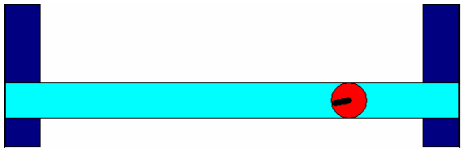
dnx, dny, dnOffsetX, dnOffsetY: LREAL

Standardized x- and y-position [0..1] and offset (for visualization).

ratio: LREAL

Ratio of x-interval and y-interval (for visualization).

GantryCutter



The modules SMC_TRAFO<n>_Gantry<n> also exist as SMC_TRAFO<n>_GantryCutter<n>. These function blocks do transformations for portal systems with one rotation axis, which is controlled in a way, that it points along the current path tangent. As additional inputs they get the rotation axis (DriveR), which must be defined as rotary axis with period 360, an offset (dOffsetX) and the direction of rotation (iDirectionR). The module for inverse transformation additionally needs the vector of the current path tangent (v), which is an output of the Interpolator.

SMC_TRAFOV_Gantry

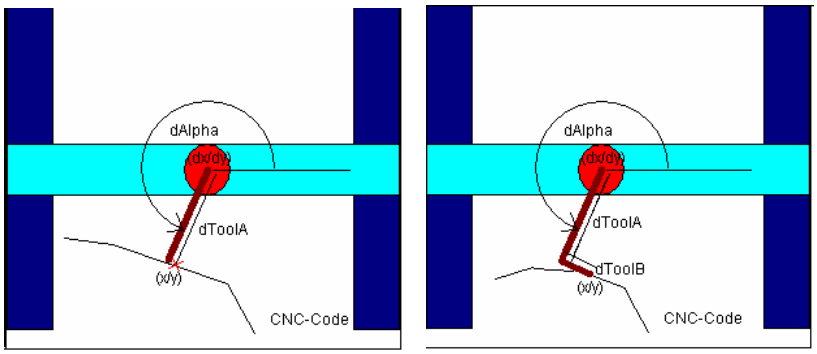
Some of the above described reverse transformations are available in a version, in which also the path velocity and path direction are used as a control variable for the axes. Those start with „SMC_TRAFOV_“ instead of „SMC_TRAFO_“. As additional inputs they need the path tangent (v) and path velocity (dVel) from the Interpolator. Besides the set positions they also show the set velocities (dvx/dvy/dvz). The advantages of this method is that the lag error in the drive can be minimized by doing an anticipatory control of the velocity – provided the drive is supporting this mode. For this reason each axis should be controlled by the SMC_ControlAxisByPosVel module.

8.2.2 Portal Systems with Tool Offset

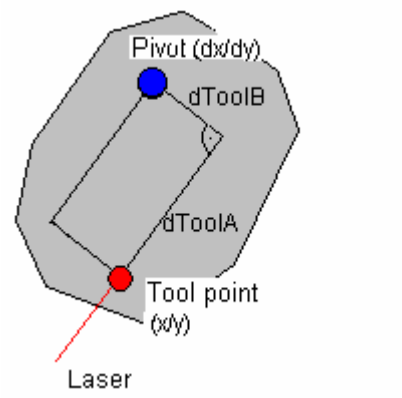
There are machine portals, whose xyz-position does not match with the tool mesh point, because this is not positioned axially in the z-axis, but translated (offset). If the z-axis cannot be rotated, this is a constant xy-offset, which can be fed as such into the standard Gantry-transformation.

However if a rotation axis by z is involved, this will not be constant offset, in this case the offset will depend on the position of the C-axis.

You must differentiate whether the tool is to be approximated as a line (if the vector between tool mesh point and axis and the scheduled alignment of the tool are matching) (->SMC_TRAFO_Gantry2Tool1) or as a parallelogram resp. a rectangular triangle (SMC_TRAFO_Gantry2Tool2):



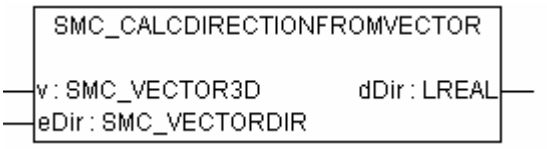
In the example shown in the following figure the tool cannot be approximated as a line, but must be approximated as a rectangular triangle:



In principle instead of the one-dimensional transformation you can also modulate the path with tool translation (tool can be approximated as a line), appropriately via SMC_Toolcorr. The difference between the two methods is the velocity of the tool point. When you are using the modulation via SMC_ToolCorr, the velocity of the rotation point is controlled according to the presettings made in the CNC program (F, E) (whereby the velocity of the tool point can vary). When you are using the one-dimensional transformation, the velocity of the tool point is determined by the CNC program.

For the calculation of the tool's orientation (dAlpha) the following function can be used:

SMC_CalcDirectionFromVector



SMC_VECTOR3D

Input vector v typically is the output vecActTangent of the Interpolator.

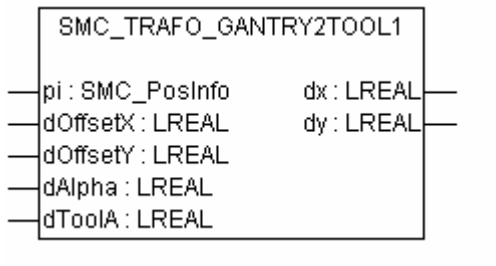
eDir: SMC_VECTORDIR

Input eDir specifies whether the direction should be calculated at a tangent to the path (SMC_tangential), or oppositely (SMC_opp_tangential) or orthogonally to the path (SMC_orthogonal_r (right to the path tangent) resp. SMC_orthogonal_l (left to the path tangent)).

dDir: LREAL

Output dDir is in angle degrees and remains constant for phases in which the Interpolator stands (v ist null-vector). eDir mostly is used as scheduled value (SMC_ControlAxisByPos) for the directional axis and as input dAlpha for the transformation.

SMC_TRAFO_Gantry2Tool1



pi: SMC_PosInfo

Target position vector. Output of the Interpolator.

dOffsetX, dOffsetY: LREAL

Offset for x- and y-axis.

dAlpha: LREAL

Orientation of the toll in angle degrees.

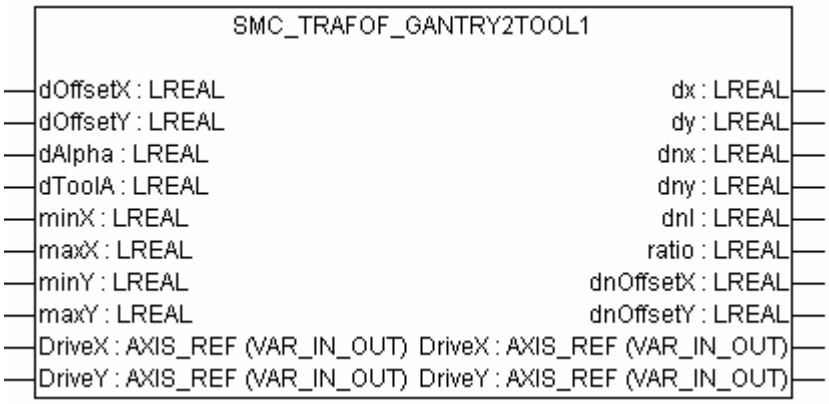
dToolA: LREAL

Length of the tool; Distance between pivot and tool point.

dx, dy: LREAL

Scheduled values for x- and y-axis.

SMC_TRAFOF_Gantry2Tool1



dOffsetX, dOffsetY: LREAL

Offset for x- and y-axis. Same values as for SMC_TRAFO_Gantry2.

dAlpha: LREAL

Orientation of the tool in angle degrees.

dToolA: LREAL

Length of the tool; Distance between pivot and tool point.

minX, maxX, minY, maxY: LREAL

Motion range (for visualization).

DriveX, DriveY: AXIS_REF

x-, y-axis.

dx, dy: LREAL

x-, y-position in GEO coordinates.

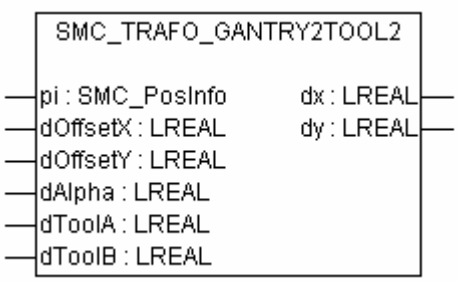
dnx, dny, dnl, dnOffsetX, dnOffsetY: LREAL

standardised x- and y-position [0..1], tool length and offsets (for visualization).

ratio: LREAL

x-interval / y-interval ration (for visualization).

SMC_TRAFO_Gantry2Tool2



pi: SMC_PosInfo

Target position vector. Output of the Interpolator.

dOffsetX, dOffsetY: LREAL

Offset for x- and y-axis.

dAlpha: LREAL

Orientation of the tool in angle degrees.

dToolA, dToolB: LREAL

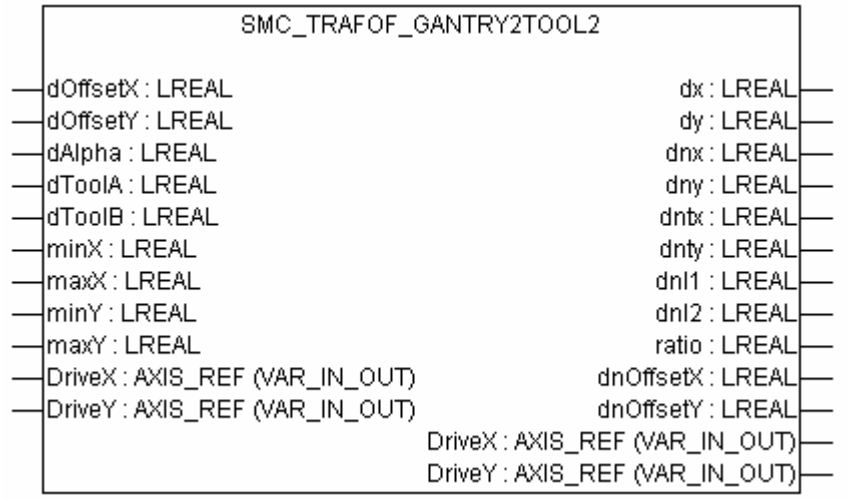
Leg length of the rectangular triangle which is between the pivot and the tool point. dToolA is the length of the leg at a tangent of the path, dToolB is the length of the leg which is orthogonal to the path.

If dToolB is positive, the tool point (x/y) is shifted to the left referring to the tool orientation, otherwise to the right.

dx, dy: LREAL

Scheduled values for x- and y-axis.

SMC_TRAFOF_Gantry2Tool2



dOffsetX, dOffsetY: LREAL

Offset for x- and y-axis. Same values as for SMC_TRAFO_Gantry2.

dAlpha: LREAL

Orientation of the tool in angle degrees.

dToolA, dToolB: LREAL

Length of the tool; Distance between pivot and tool point.

minX, maxX, minY, maxY: LREAL

Motion range (for visualization).

DriveX, DriveY: AXIS_REF

x-, y-axis.

dx, dy: LREAL

x-, y-position in GEO-coordinates.

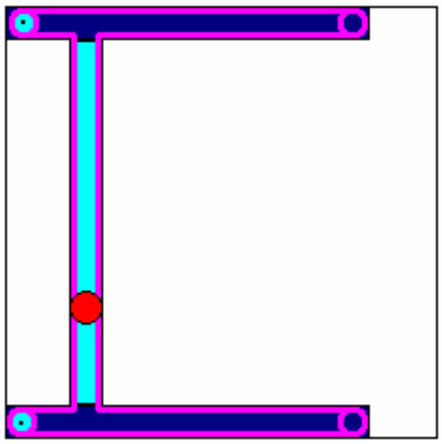
dnx, dny, dnl1, dnl2, dnOffsetX, dnOffsetY: LREAL

standardised x- and y-position [0..1], tool length and offsets (for visualization).

ratio: LREAL

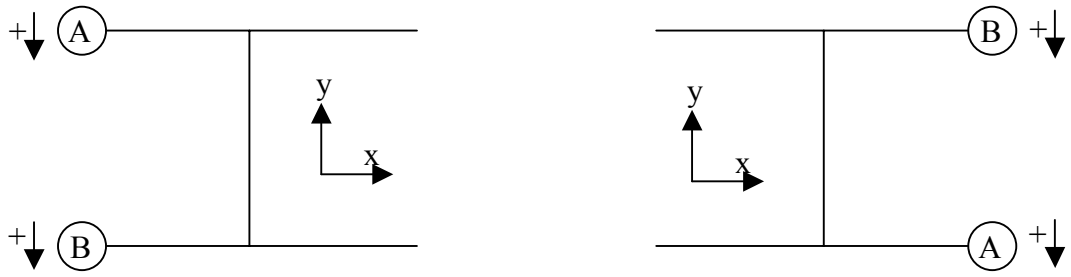
x-interval /y-interval ration (for visualization).

8.2.3 H-Portal-System with stationary drives



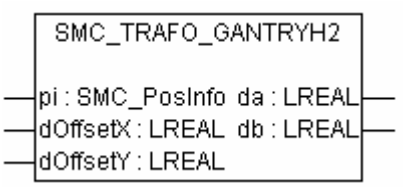
This kinematic system is similar to those described above, but the drives are mounted stationary and move the sledge and the y-axis over a multi-turned-round belt (displayed pink-colored in the picture).

The transformation fits fort he following drive configurations; other configurations can be reached by interchanging x and y:



Please regard, that for this transformation a special reference move is necessary. If you want a move in y-direction, the drives A and B must be moved in parallel.; for a pure x-move they have to be counter rotated. If the reference position has been found, the x- and y-values calculated by the forward transformation FB are used as offset (dOffsetX and dOffsetY).

SMC_TRAFO_GantryH2



pi: SMC_PosInfo

Target position vector. Output of the Interpolator

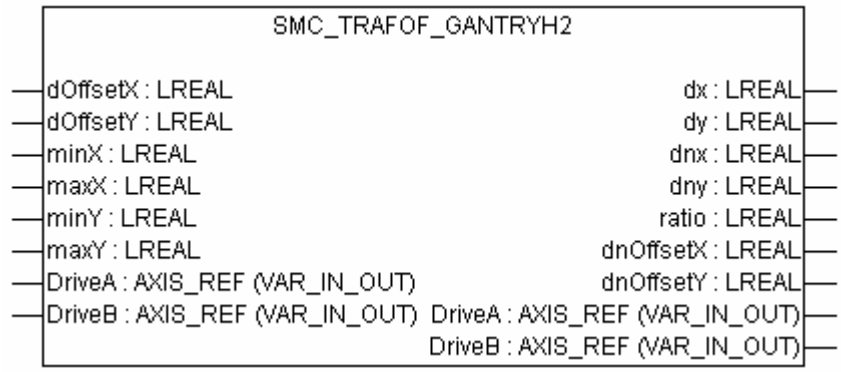
dOffsetX, dOffsetY: LREAL

Offset for x- and y-axis.

da, db: LREAL

Target values for A- and B-axis.

SMC_TRAFOF_GantryH2



dOffsetX, dOffsetY: LREAL

Offset for x- and y-axis. Same values as described for SMC_TRAFO_GantryH2.

minX, maxX, minY, maxY: LREAL

Move range (for visualization).

DriveA, DriveB: AXIS_REF

A-, B-axis.

dx, dy: LREAL

x-, y-position in Geo-coordinates.

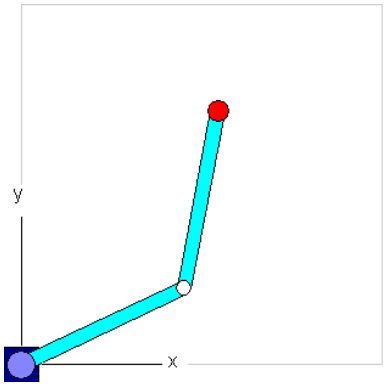
dnx, dny, dnOffsetX, dnOffsetY: LREAL

Standardized x- and y-position [0..1] and offset (for visualization).

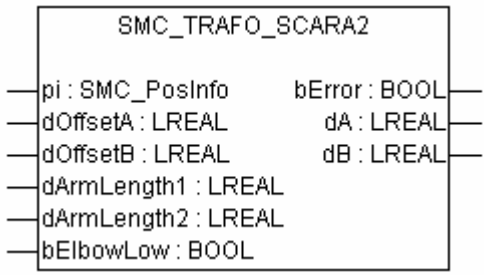
ratio: LREAL

Ratio of x-interval and y-interval (for visualization).

8.2.4 2-Jointed Scara-Systems



SMC_TRAFO_Scara2



pi: SMC_PosInfo

Target position vector. Output of the Interpolator.

dOffsetA, dOffsetB: LREAL

Offset for A- and B-axis.

dArmLength1, dArmLength2: LREAL

Length of first and second arm.

bElbowLow: BOOL

Elbow down (TRUE) or top (FALSE)

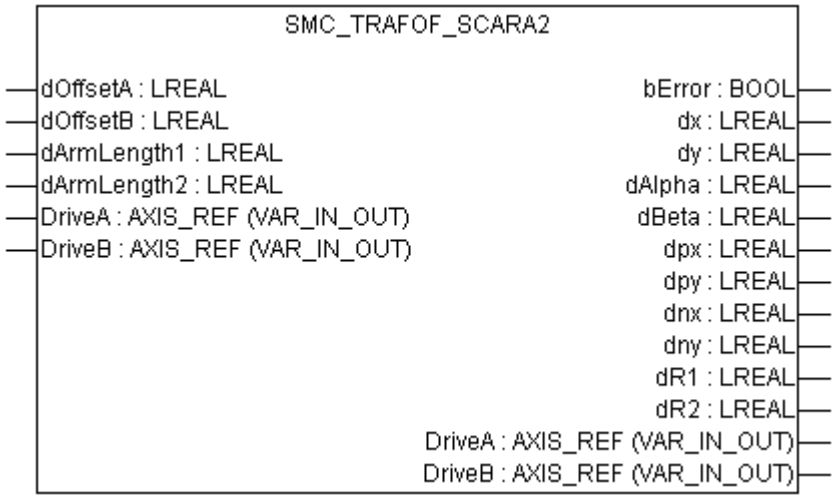
bError BOOL

TRUE: Invalid values.

dA, dB: LREAL

Axis position A- resp. B-axis.

SMC_TRAFOF_Scara2



dOffsetA, dOffsetB: LREAL

Offset for A- and B-axis. Identical values like at SMC_TRAFO_Scara2.

dArmLength1, dArmLength2: LREAL

Length of first and second arm.

DriveA, DriveB: AXIS_REF

A-, B-axis.

bError: BOOL

TRUE: Invalid values.

dx, dy: LREAL

x-, y-position in geo-coordinates.

dAlpha, dBeta: LREAL

Joint angle (axis positions without offset). (for visualization)

dpx, dpy: LREAL

Standardized position of the first joint]-1..1[(for visualization)

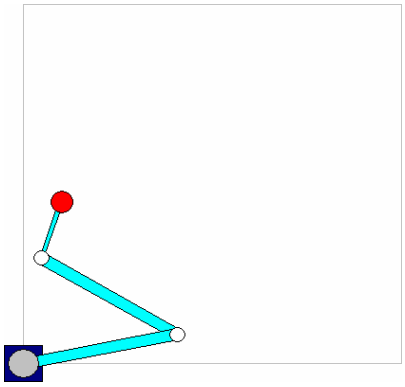
dnx, dny: LREAL

Standardized position of the manipulator.]-1..1[(for visualization)

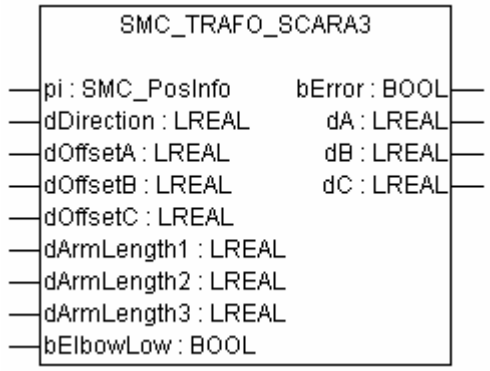
dR1, dR2: LREAL

Relative arm lengths. dR1+dR2 = 1.]0..1[(for visualization)

8.2.5 3-Jointed Scara-Systems



SMC_TRAFO_Scara3



pi: SMC_PosInfo

Target position vector. Output of the Interpolator.

dDirection: LREAL

Direction angle of the last joint in degrees. (0° W, 90° N)

dOffsetA, dOffsetB, dOffsetC: LREAL

Offset for A-, B- and C-axis.

dArmLength1, dArmLength2, dArmLength3: LREAL

Length of the arms.

bElbowLow: BOOL

Elbow (1. and 2. joint) down (TRUE) resp. top (FALSE)

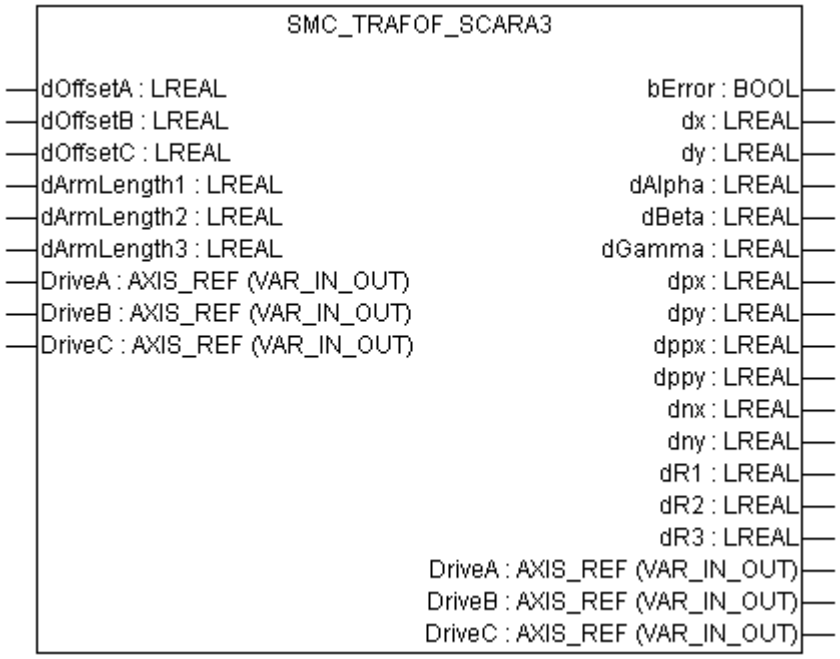
bError: BOOL

TRUE: Invalid values.

dA, dB, dC: LREAL

Axis position A-, B-, C-axis.

SMC_TRAFOF_Scara3



dOffsetA, dOffsetB, dOffsetC: LREAL

Offset for A-, B- and C-axis. Same values as described for SMC_TRAFO_Scara3.

dArmLength1, dArmLength2, dArmLength3: LREAL

Lenght of the arms.

DriveA, DriveB, DriveC: AXIS_REF

A-, B- and C-axis.

bError: BOOL

TRUE: Invalid values.

dx, dy: LREAL

x-, y-Position in Geo-coordinates.

dAlpha, dBeta, dGamma: LREAL

Joint angle (axis positions without offset). (for visualization)

dpx, dpy: LREAL

Standardized position of the first joint]-1..1[(for visualization)

dppx, dppy: LREAL

Standardized position of the second joint]-1..1[(for visualization)

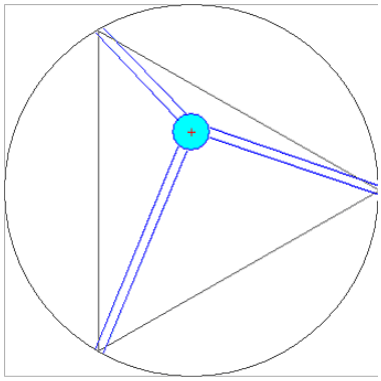
dnx, dny: LREAL

Standardized position of the manipulator.]-1..1[(for visualization)

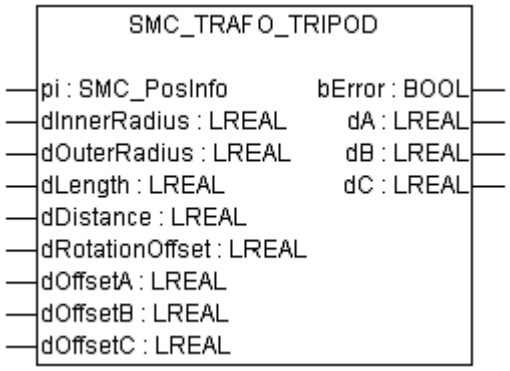
dR1, dR2, dR3: LREAL

Relative arm lengths. dR1+dR2+dR3 = 1.]0..1[(for visualization)

8.2.6 Parallel Kinematics



SMC_TRAFO_Tripod



pi: SMC_PosInfo

Target position vector. Position of the centre of the inner ring. Output of the Interpolator.

dInnerRadius: LREAL

Radius of the inner ring.

dOuterRadius: LREAL

Radius of the outer ring.

dLength: LREAL

Strut lengths.

dDistance: LREAL

Distance between two connected struts at the outer and inner ring.

dRotationOffset: LREAL

Position of axis A in angular degrees (mathematical sense) in relating to the origin (0/0).

dOffsetA, dOffsetB, dOffsetC: LREAL

Offset of the particular axes.

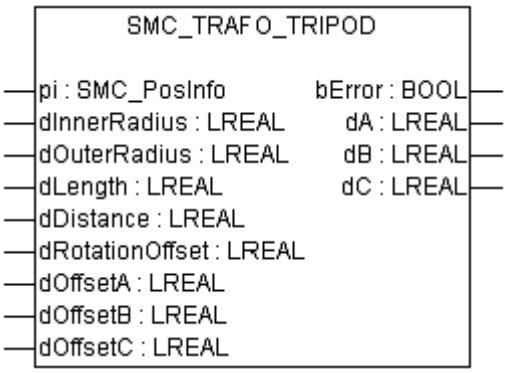
bError BOOL

Offset of the particular axes.

dA, dB, dC: LREAL

Axis position A-, B-, C-axis.

SMC_TRAFO_Tripod



dInnerRadius, dOuterRadius, dLength, dDistance, dRotationOffset, dOffsetA, dOffsetB, dOffsetC: LREAL

see SMC_TRAFO_TRIPOD

DriveA, DriveB, DriveC: AXIS_REF

A-, B- and C-axis.

bError: BOOL

TRUE: Invalid values.

dx, dy, dz: LREAL

x-, y-, z-Position of the centre of the inner ring in geo-coordinates.

dnx, dny: LREAL

Standardized position of the manipulator. (for visualization)

dRatioInnerOuter: LREAL

Ration of the radius of the inner ring to that of the outer ring. (for visualization)

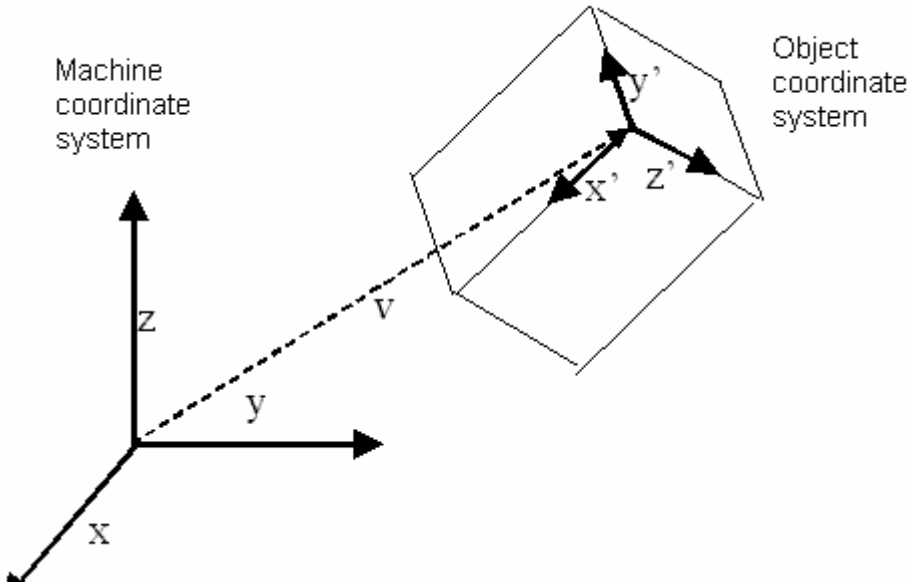
adnxi, adnyi, adnxo, adnyo: ARRAY[0..5] OF LREAL

Standardized start and end position of the bars. (for visualization)

8.3 Spacial Transformations

In some cases the coordinate systems of the machine and the work piece differ from each other. The transformation modules described above transform Cartesian coordinates of the work piece to machine coordinates of the work piece. However previously it might be necessary (if the work piece is not orientated exactly according to the CNC program), that the path coordinates calculated by the Interpolator must be transformed before getting handed over to the machine transformation.

Imagine an usual portal (X/Y/Z). The tool point of the portal must be moved on the surface of the work piece which is positioned transversely in the space:



There are several possibilities to describe the reference between two coordinate systems. A coordinate transformation always is composed of a spacial translation and a spacial rotation. The translation is described by a three-dimensional vector, the rotation is either described by three angles (e.g. YawPitchRoll) or by the three unit vectors of the new (Object-)coordinate system x', y', z'.

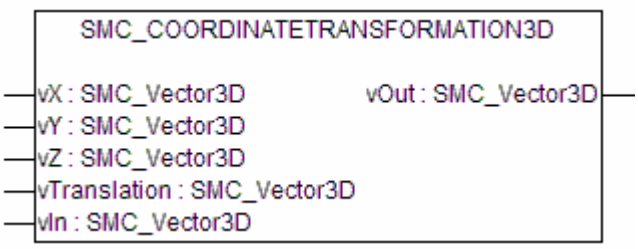
If the method of the three rotation angles is chosen, those e.g. can be defined according to the Roll/Pitch/Yaw (RPY) convention. In this case the new coordinate system results from the old one by a rotation around some axes. You can imagine the RPY (α, β, γ) move in two ways, which both provide the same result:

1. Setting out from coordinate system (x,y,z) the coordinate system is rotated around the z-axis by angle γ in mathematically positive direction. This will result in the new coordinate system (x1, y1, z1=z). Now fix axis y1 of the new coordinate system and rotate the coordinate system β, thus generating (x2, y2=y1, z2). Finally rotate this resulting coordinate system around x2 by angle α. Thus you receive (x'=x2, y', z').

2. Setting out from coordinate system (x,y,z) rotate the coordinate system around the x-axis by α. Then rotate the resulting coordinate system (xa=x, ya, za) around the original y-axis (not ya!!) by β (xb, yb, zb) and subsequently around the original z-axis by γ, by what (x', y', z') will result.

SMC_CoordinateTransformation
3D

This module calculates the coordinates of a position (existing in the old coordinate system) referring to the new coordinate system. For this purpose the coordinate transformation of the new resp. old coordinate system is preset via translation vector and the new unit vectors.



vX, vY, vZ: SMC_Vector3D

Unit vectors of the new coordinate system referring to the old one.

vTranslation: SMC_Vector3D

Translation vector. Vector from the old coordinate origin to the new origin referring to the old coordinate system.

vIn: SMC_Vector3D

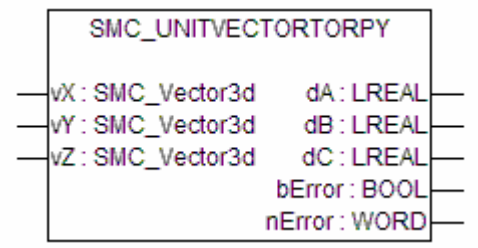
Position to be transformed.

vOut: SMC_Vector3D

Transformed position.

SMC_UnitVectorToRPY

Module for calculation of the RPY-angle (referring to the old coordinate system) from the unit vectors of the new coordinate system.



vX, vY, vZ: SMC_Vector3D

Unit vectors of the new coordinate system referring to the old one.

dA, dB, dC: LREAL

RPY-angle in radian measure.

bError: BOOL

Invalid input values.

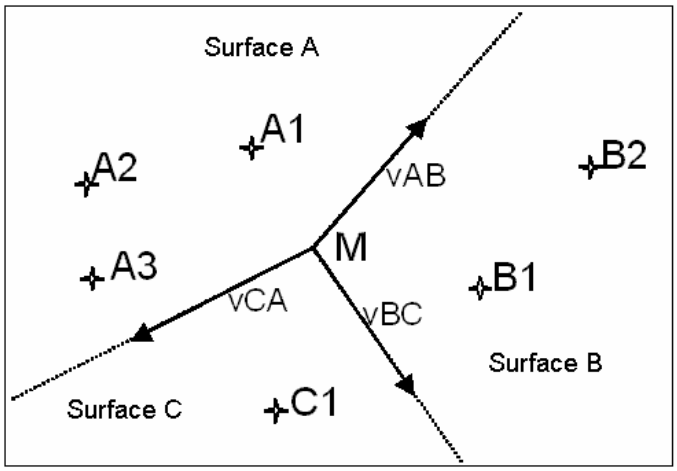
nError: WORD

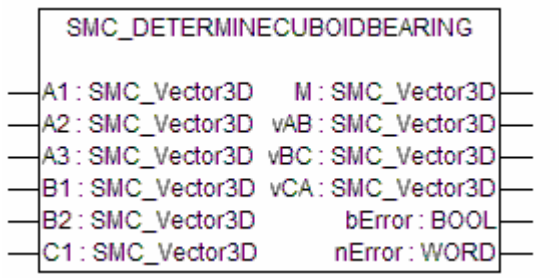
Error description:

0	No error (bError = FALSE)
1	Vectors do not have length 1
2	Vectors are not perpendicular to each other
3	No right hand system

SMC_DetermineCuboidBearing

Module for determination of the position of a cuboid (corner mark, edge alignment) in the space by the presetting of 6 (3/2/1) points:





A1, A2, A3: SMC_Vector3D

Three points on a end surface A of a cube, which may not lie on a line.

B1, B2: SMC_Vector3D

Two points on another end surface of the cube, whose projection on surface A may not be identical.

C1: SMC_Vector3D

Points on a further end surface of the cube.

M: SMC_Vector3D

Corner point of the cube.

vAB, vBC, vCA: SMC_Vector3D

Unit vectors on the edge lines of the cube.

bError: BOOL

Invalid input values.

nError: WORD

Error description:

0	No error (bError = FALSE)
1	A1, A2, A3 lie on a line
2	Projections of B1 and B2 on surface A are identic

9 The Library SM_Error.lib

This library must be available in each project, because it contains all error definitions. It is used to display each error produced by a SoftMotion module as a string.

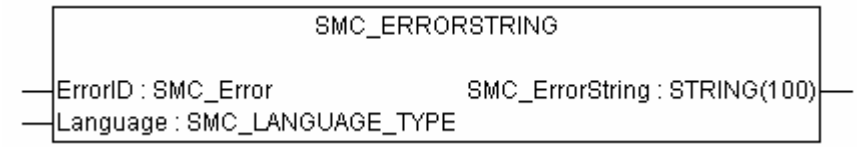
Basically the application programmer should regard that two types of error might occur in his program. On the one hand there might be **Drive errors**, that are errors in the drive (e.g. lag errors, missing power supply etc.). On the other hand there are **Module errors**, which are errors reported by modules via the outputs *Error* und *ErrorID*, and which often are caused by an incorrect parameterizing.

Drive errors must be read via MC_ReadAxisError resp. MC_ReadParameter and where appropriate deleted via MC_Reset. Drive errors are drive specific and not standardized.

Module errors can be converted via the functions of the SM_Error.lib to strings. As these errors can occur at all SoftMotion modules and would have to be gathered by the application, an additional functionality has been implemented in the AXIS_REF data structure, always saving the last error which has occurred at a module with an AXIS_REF input. Via output **FBErrorOccured** of MC_ReadStatus or via function block **SMC_ReadFBError** it can be checked whether resp. which module error has occurred at last. Via function SMC_**ClearFBError** that error can be deleted.

9.1 Function blocks

9.1.1 SMC_ErrorString



Depending on the inputs ErrorID (SMC_Error) and Language (SMC_LANGUAGE_TYPE (english, german)) the function SMC_ErrorString returns a string representing the error.

9.2 The enumeration SMC_Error

The enumeration SMC_Error contains all error numbers, which might be returned by SoftMotion function blocks:

Error no.	Module	Enum value	Description
0	all	SMC_NO_ERROR	no error
10	DriveInterface	SMC_DI_SWLIMITS_EXCEEDED	Position outside of permissible range (SWLimit)
20	all motion generating modules	SMC_REGULATOR_OR_START_NOT_SET	Controller enable not done or brake applied
30	DriveInterface	SMC_FB_WASNT_CALLED_DURING_MOTION	Motion-creating module has not been called again before end of the motion.

Error no.	Module	Enum value	Description
31	All modules	SMC_AXIS_IS_NO_AXIS_REF	Given AXIS_REF variable is not of type AXIS_REF
32	All motion generating modules	SMC_AXIS_REF_CHANGED_DURING_OPERATION	The fed in AXIS_REF-variable has been exchanged while the module was active
50	SMC_Homing	SMC_3SH_INVALID_VELACC_VALUES	invalid velocity or acceleration values
51	SMC_Homing	SMC_3SH_MODE_NEEDS_HWLIMIT	Mode requests (for safety reasons) the use of the end switches
70	SMC_SetControllerMode	SMC_SCM_NOT_SUPPORTED	Mode not supported
75	SMC_SetTorque	SMC_ST_WRONG_CONTROLLER_MODE	Axis is not in correct controller mode
80	SMC_ResetAxisGroup	SMC_RAG_ERROR_DURING_STARTUP	Error at startup of the axisgroup
90	SMC_ChangeGearingRatio	SMC_CGR_ZERO_VALUES	invalid values
91	SMC_ChangeGearingRatio	SMC_CGR_DRIVE_POWERED	Gearing parameters may not be changed as long as the drive is under control
110	MC_Power	SMC_P_FTASKCYCLE_EMPTY	The axis does not contain any information on the cycle time (fTaskCycle = 0)
120	MC_Reset	SMC_R_NO_ERROR_TO_RESET	Axis without error
121	MC_Reset	SMC_R_DRIVE_DOESNT_ANSWER	Axis does not perform error-reset.
122	MC_Reset	SMC_R_ERROR_NOT_RESETTABLE	Error could not be reset
123	MC_Reset	SMC_R_DRIVE_DOESNT_ANSWER_IN_TIME	Communication with the axis did not work
130	MC_ReadParameter, MC_ReadBoolParameter	SMC_RP_PARAM_UNKNOWN	Parameter number unknown
131	MC_ReadParameter, MC_ReadBoolParameter	SMC_RP_REQUESTING_ERROR	Error during transmission to the drives; see error number in FB instance ReadDriveParameter (SM_DriveBasic.lib)
140	MC_WriteParameter, MC_WriteBoolParameter	SMC_WP_PARAM_INVALID	Parameter number unknown or writing not allowed
141	MC_WriteParameter, MC_WriteBoolParameter	SMC_WP_SENDING_ERROR	See error number in module instance WriteDriveParameter (Drive_Basic.lib)

Error no.	Module	Enum value	Description
170	MC_Home	SMC_H_AXIS_WASNT_STANDSTILL	Axis has not been in standstill state
171	MC_Home	SMC_H_AXIS_DIDNT_START_HOMING	Error at start of Homing-action
172	MC_Home	SMC_H_AXIS_DIDNT_ANSWER	Communication error
173	MC_Home	SMC_H_ERROR_WHEN_STOPPING	Error at stop after Homing. Deceleration set?
180	MC_Stop	SMC_MS_UNKNOWN_STOPPING_ERROR	Unknown error at stop
181	MC_Stop	SMC_MS_INVALID_ACCDEC_VALUES	Invalid velocity or acceleration values
182	MC_Stop	SMC_MS_DIRECTION_NOT_APPLICABLE	Direction=shortest not applicable
183	MC_Stop	SMC_MS_AXIS_IN_ERRORSTOP	Drive is in errorstop status. Stop cannot be executed.
201	MC_MoveAbsolute	SMC_MA_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
202	MC_MoveAbsolute	SMC_MA_INVALID_DIRECTION	Direction error
226	MC_MoveRelative	SMC_MR_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
227	MC_MoveRelative	SMC_MR_INVALID_DIRECTION	Direction error
251	MC_MoveAdditive	SMC_MAD_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
252	MC_MoveAdditive	SMC_MAD_INVALID_DIRECTION	Direction error
276	MC_MoveSuperImposed	SMC_MSI_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
277	MC_MoveSuperImposed	SMC_MSI_INVALID_DIRECTION	Direction error
301	MC_MoveVelocity	SMC_MV_INVALID_ACCDEC_VALUES	Invalid velocity or acceleration values
302	MC_MoveVelocity	SMC_MV_DIRECTION_NOT_APPLICABLE	Direction=shortest/fastest not applicable
325	MC_PositionProfile	SMC_PP_ARRAYSIZE	Erroneous array size
326	MC_PositionProfile	SMC_PP_STEP0MS	Step time = t#0s
350	MC_VelocityProfile	SMC_VP_ARRAYSIZE	erroneous array size
351	MC_VelocityProfile	SMC_VP_STEP0MS	Step time = t#0s
375	MC_AccelerationProfile	SMC_AP_ARRAYSIZE	erroneous array size

Error no.	Module	Enum value	Description
376	MC_AccelerationProfile	SMC_AP_STEP0MS	Step time = t#0s
400	MC_TouchProbe	SMC_TP_TRIGGEROCCUPIED	Trigger already active
401	MC_TouchProbe	SMC_TP_COULDNT_SET_WINDOW	DriveInterface does not support the window function
402	MC_TouchProbe	SMC_TP_COMM_ERROR	Communication error
410	MC_AbortTrigger	SMC_AT_TRIGGERNOTOCCUPIED	Trigger already de-allocated
500	SMC_ControlAxisByPos	SMC_CAP_GAP_VELACCDEC_INVALID	Invalid velocity or acceleration values
510	SMC_ControlAxisByPosVel	SMC_CAPV_GAP_VELACCDEC_INVALID	Invalid velocity or acceleration values
520	SMC_ControlAxisByVel	SMC_CAV_GAP_VELACCDEC_INVALID	Invalid velocity or acceleration values
600	SMC_CamRegister	SMC_CR_NO_TAPPETS_IN_CAM	CAM does not contain any tappets
601	SMC_CamRegister	SMC_CR_TOO_MANY_TAPPETS	Tappet-GroupID exceeds MAX_NUM_TAPPETS
602	SMC_CamRegister	SMC_CR_MORE_THAN_32_ACCESSES	more than 32 accesses on one CAM_REF
625	MC_CamIN	SMC_CI_NO_CAM_SELECTED	no CAM selected
626	MC_CamIN	SMC_CI_MASTER_OUT_OF_SCALE	Master axis out of valid range
627	MC_CamIn	SMC_CI_RAMPIN_NEEDS_VELACC_VALUES	for the ramp_in function velocity and acceleration values must be specified
628	MC_CamIn	SMC_CI_SCALING_INCORRECT	Scaling variables fEditor/TableMasterMin/Max are not correct
675	MC_GearIn	SMC_GI_RATIO_DENOM	RatioDenominator = 0
676	MC_GearIn	SMC_GI_INVALID_ACC	Acceleration invalid
677	MC_GearIn	SMC_GI_INVALID_DEC	Deceleration invalid
725	MC_Phase	SMC_PH_INVALID_VELACCDEC	Velocity, Deceleration- or Acceleration values invalid
726	MC_Phase	SMC_PH_ROTARYAXIS_PERIOD0	Rotation axis with fPositionPeriod = 0
750	All modules using MC_CAM_REF as input	SMC_NO_CAM_REF_TYPE	Given CAM is not of type MC_CAM_REF

Error no.	Module	Enum value	Description
1001	SMC_Interpolator	SMC_INT_VEL_ZERO	Path not unfahrbar, da Soll-Geschwindigkeit = 0.
1002	SMC_Interpolator	SMC_INT_NO_STOP_AT_END	Last path object has *Vel_End* > 0.
1003	SMC_Interpolator	SMC_INT_DATA_UNDERRUN	GEOINFO-List processed in DataIn, but end of list not set. Reason: Forgotten to set *EndOfList* of the queue in *DataIn* or SMC_Interpolator is faster than the path generating modules.
1004	SMC_Interpolator	SMC_INT_VEL_NONZERO_AT_STOP	Velocity at Stop > 0.
1005	SMC_Interpolator	SMC_INT_TOO_MANY_RECURSIONS	To much SMC_Interpolator recursions. SoftMotion-Error.
1006	SMC_Interpolator	SMC_INT_NO_CHECKVELOCITIES	Input-OutQueue DataIn has not as last processed module SMC_CHeckVelocities
1007	SMC_Interpolator	SMC_INT_PATH_EXCEEDED	Internal / numeric error
1050	SMC_Interpolator2Dir	SMC_INT2DIR_BUFFER_TOO_SMALL	Data buffer too small
1051	SMC_Interpolator2Dir	SMC_INT2DIR_PATH_FITS_NOT_IN_QUEUE	Path does not go completely in queue
1080	SMC_Interpolator	SMC_WAR_INT_OUTQUEUE_TOO_SMALL	Warning: OutQueue *DataIn* dimensioned too small. Meeting of stops cannot be guaranteed.
1081	SMC_Interpolator	SMC_WAR_END_VELOCITIES_INCORRECT	Warning: End velocities inconsistent.
1100	SMC_CheckVelocities	SMC_CV_ACC_DEC_VEL_NONPOSITIVE	Velocity, Deceleration- or Acceleration values impermissible.
1200	SMC_NCDecoder	SMC_DEC_ACC_TOO_LITTLE	Acceleration value impermissible.
1201	SMC_NCDecoder	SMC_DEC_RET_TOO_LITTLE	Acceleration value impermissible.
1202	SMC_NCDecoder	SMC_DEC_OUTQUEUE_RAN_EMPTY	Data underrun. Queue has been read and is empty.
1500	All function blocks using SMC_CNC_REF	SMC_NO_CNC_REF_TYPE	The given CNC program is not of type SMC_CNC_REF
1501	All function blocks using SMC_OUTQUEUE	SMC_NO_OUTQUEUE_TYPE	The given OutQueue is not of type SMC_OUTQUEUE
2000	SMC_ReadNCFile	SMC_RNCF_FILE_DOESNT_EXIST	File does not exist
2001	SMC_ReadNCFile	SMC_RNCF_NO_BUFFER	No buffer allocated

Error no.	Module	Enum value	Description
2002	SMC_ReadNCFile	SMC_RNCF_BUFFER_TOO_SMALL	Buffer too small
2003	SMC_ReadNCFile	SMC_RNCF_DATA_UNDERRUN	Data underrun. Buffer has been read, is empty.
2004	SMC_ReadNCFile	SMC_RNCF_VAR_COULDNT_BE_REPLACED	placeholder variable could not be replaced
2050	SMC_ReadNCQueue	SMC_RNCQ_FILE_DOESNT_EXIST	File could not be opened.
2051	SMC_ReadNCQueue	SMC_RNCQ_NO_BUFFER	no buffer defined.
2052	SMC_ReadNCQueue	SMC_RNCQ_BUFFER_TOO_SMALL	Buffer too small.
2053	SMC_ReadNCQueue	SMC_RNCQ_UNEXPECTED_EOF	unexpected end of file.
2100	SMC_AxisDiagnosticLog	SMC_ADL_FILE_CANNOT_BE_OPENED	File could not be opened
2101	SMC_AxisDiagnosticLog	SMC_ADL_BUFFER_OVERRUN	Buffer-overrun; WriteToFile must be called more frequently
2200	SMC_ReadCAM	SMC_RCAM_FILE_DOESNT_EXIST	File could not be opened.
2201	SMC_ReadCAM	SMC_RCAM_TOO_MUCH_DATA	saved CAM to big.
2202	SMC_ReadCAM	SMC_RCAM_WRONG_COMPILE_TYPE	wrong compilation mode
2203	SMC_ReadCAM	SMC_RCAM_WRONG_VERSION	File has wrong version
2204	SMC_ReadCAM	SMC_RCAM_UNEXPECTED_EOF	unexpected end of file
3001	SMC_WriteDriveParamsToFile	SMC_WDPF_CHANNEL_OCCUPIED	Parameter selection channel is occupied
3002	SMC_WriteDriveParamsToFile	SMC_WDPF_CANNOT_CREATE_FILE	File could not be created
3003	SMC_WriteDriveParamsToFile	SMC_WDPF_ERROR_WHEN_READING_PARAMS	Error at reading of the parameters
3004	SMC_WriteDriveParamsToFile	SMC_WDPF_TIMEOUT_PREPARING_LIST	Timeout during preparing the parameter list
5000	SMC_Encoder	SMC_ENC_DENOM_ZERO	Nominator of the conversion factor (dwRatioTechUnits Denom) of the Encoder reference is 0.
5001	SMC_Encoder	SMC_ENC_AXISUSEDBYOTHERFB	Other module trying to process motion on the Encoder axis

10 The library SM_FileFBs.lib

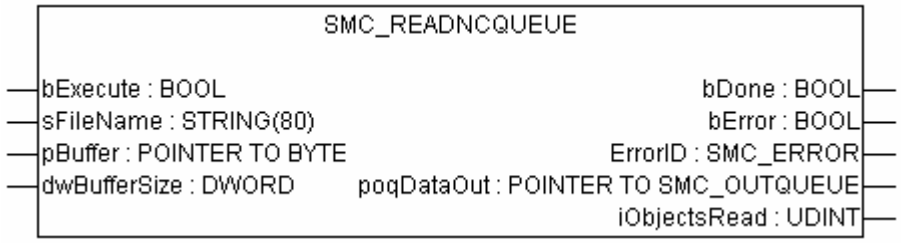
10.1 Overview

This library provides modules for the file functionality. They can only be used, if the 3S system libraries SysLibFile.lib and Standard.lib are also available.

10.2 CNC function blocks

SMC_ReadNCQueue

This module reads an OutQueue file, which has been created by the CNC editor (see chapter 3.3), from the PLC file system and provides an OutQueue structure, which typically is processed by the Interpolator.



Inputs of the module:

bExecute: BOOL

At a rising edge the module starts with reading the queue.

sFileName: STRING(80)

File path and - name.

pBuffer: POINTER TO BYTE

Pointer on a sufficiently large, free data area (buffer) which is allocated in the IEC application.

dwBufferSize: DWORD

Size of the buffer in Byte.

Outputs of the module:

bDone: BOOL

Gets set after the queue has been read completely.

bError: BOOL

TRUE: Error has occurred.

ErrorID: SMC_ERROR

Error number.

poqDataOut: POINTER TO SMC_OUTQUEUE

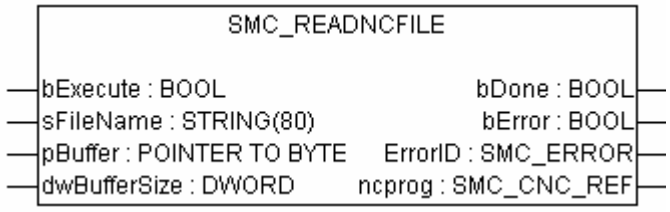
Pointer on a queue which has been read in.

iObjectsRead: UDINT

Number of the SMC_GeoInfo objects which have been read and written to the queue up to now.

SMC_ReadNCFile

This module reads a NC-ASCII-file from the file system of the controller, in order to make it available for the SMC_NCDecoder. Thus at runtime a NC program can be read in and realized.



Inputs of the function block:

bExecute: BOOL

At a rising edge the module starts to read in the program.

sFileName: STRING(80)

File path and name.

pvl: POINTER TO SMC_VARLIST

Pointer on a SMC_VARLIST object (see below). If no variables are used in the CNC program, this input will not be used.

pBuffer: POINTER TO BYTE

Pointer to a free data area which is allocated in the IEC application and which is big enough.

dwBufferSize: DWORD

Size of the data area in Bytes.

Outputs of the function block:

bDone: BOOL

Is set as soon as the program has been read in completely.

bError: BOOL

TRUE: Error occurred.

ErrorID: SMC_ERROR

Error number.

bExecuteDecoder: BOOL

Signal, which should trigger the input Execute of the SMC_NCDecoder module.

ncprog: SMC_CNC_REF

CNC program. Input of the succeeding SMC_NCDecoder module.

SMC_VARLIST structure

The standard IEC1131-3 does not describe a possibility to acquire the value of a variable from its symbolic name, which e.g. is be available as a string. This however is necessary, if the variable functionality (see 3.2), which is available for the user by compile option ‚Create program variable on compile' (see 3.7), also should be available for reading in the CNC program from a file. This can be managed by using the structure SMC_VARLIST. It provides a variable *wNumberVars*, which contains the number of all used variables as well as a pointer *psvVarList* on the first element of an array of ***SMC_SingleVar***, which contains the variable description and values. A *SMC_SingleVar* object contains the string *strVarName*, which provides the name of the variable, as used in the NC program, in capital letters. Besides that the object provides the value of the variable, which can be used depending on the type as DINT (*diValue*) or REAL (*fValue*) parameter.

An example:

In the NC program which is read by using SMC_ReadNCFile from a file, there are two variables g_fTestX (REAL) and g_byCommand (BYTE):

```
N0 G$g_byCommand$ X$g_fTestX$
```

So you have to define the following variables:

```
asv: ARRAY[0..1] OF SMC_SingleVar :=
            (strVarName:='G_BYCOMMAND', diValue:=1, fValue:=1.0),
            (strVarName:='G_FTESTX', diValue:=0, fValue:=-1000.0);
vl: SMC_VarList:=(wNumberVars:=2);
```

Before calling module SMC_ReadNCFile, whose pvl-inputs then will be fed with ADR(vl), you can change the values of the variables; e.g. in order to modify g_fTestX :

```
asv[1].fValue:=1050;
```

and you must define the assignment between SMC_VarList and AMC_SingleVar:

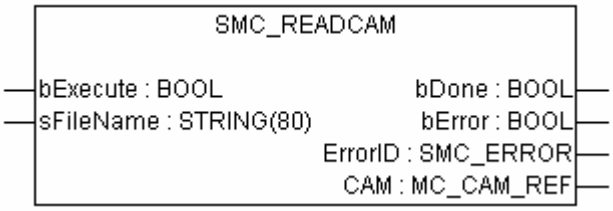
```
vl.psvVarList := ADR(asv[0]);
```

If a variable cannot be replaced, an error will be dumped and the module will abort.

10.3 CAM Function Blocks

SMC_ReadCAM

This module is used to load a CAM, which has been created in the CAM editor and has been saved in a *.CAM file (see chapter 4.4.3), at runtime and to make it available for the modules MC_CamTableSelect and MC_CamIn.



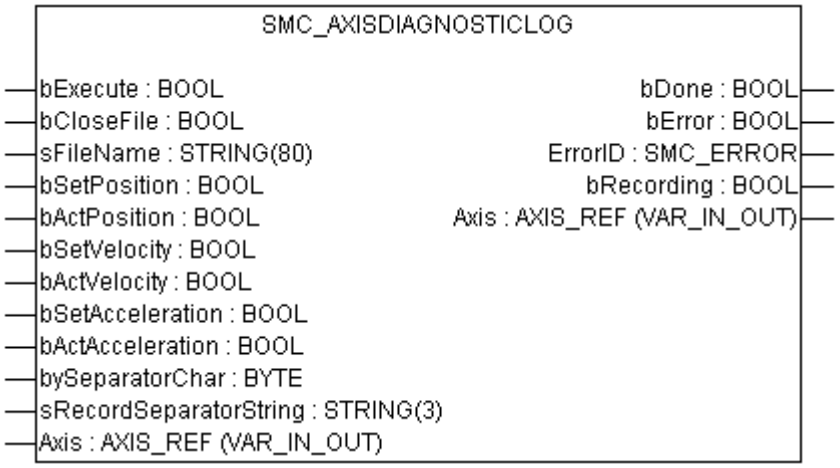
The size of a loadable CAM is limited by the global constants *gc_SMC_FILE_MAXCAMEL* (number of elements) and *gc_SMC_FILE_MAXCAMTAP* (number of CAM switch actions).

10.4 Diagnosis function blocks

SMC_AxisDiagnosticLog

This module can be used to write cyclically a selection of values of an axis to a file. A file created in this way ideally can be used for diagnosis purposes.

As the writing of data on a data medium usually needs some time, this module stores the collected data in a buffer of size 5kByte and the data will not be written until module action **WriteToFile** is called. This action call should be placed in a slower (ca. 50 ms) task of lower priority, in order to not hinder the actual motion task and not to disturb the motion behaviour. As soon as the buffer exceeds the module will create an error output.



Inputs of the module:

bExecute: BOOL

At a rising edge the module starts processing. If there is already a file with the same name, this will be overwritten.

bCloseFile: BOOL

The module closes the file as soon as this input gets TRUE.

sFileName: STRING(80)

File path and –name.

bSetPosition, bActPosition, bSetVelocity, bActVelocity, bSetAcceleration, bActAcceleration: BOOL

These inputs define whether the associated values of the axis should be written to the file.

bySeparatorChar: BYTE (Default: TAB)

ASCII-Code of the letter, which should be written between two values of same date.

sRecordSeparatorString: STRING(3) (Default: ,RN')

String, which should be written at the end of a date.

Axis: AXIS_REF;

Axis, which should be controlled.

Outputs of the module:

bDone BOOL

TRUE: Logging terminated, file closed.

bError: BOOL

TRUE: Error occurred

ErrorID: SMC_ERROR

Error number

bRecording: BOOL

TRUE: Module is recording.

11 Programming Examples

11.1 Overview

For controlling a drive hardware with a CoDeSys project and via SoftMotion, the following items have to be regarded:

- The SoftMotion functionality must be activated in the target settings, tab 'General'.
- The libraries Drive_Basic.lib and the manufacturer specific <BusInterfaceBezeichnung>Drive.lib must be included in the CoDeSys project so that the Drive Interface can be used for the communication with the drives.
- In the Drive Interface(PLC Configuration) the structure of the drive hardware must be mapped and parameterized; after a compilation of the project then automatically the appropriate global variables will be created.
- A Task configuration must be created.
- An IEC program has to be created (in a CoDeSys editor), which processes the desired movements by calling the appropriate modules. In order to have available the appropriate SoftMotion functions, the libraries SM_CNC.lib resp. SM_PLCopen.lib must be included in the CoDeSys project. In the CNC- resp. CAM-Editor multi-axis-movements resp. CAMs for the controlling of the drives can be programmed graphically and in character-based format; out of this programs CoDeSys then will automatically create the corresponding data structures (CNC Data, CAM Data), which can be accessed by the IEC program.

See the following programming examples:

- Drive Interface: Create PLC Configuration, chapter 2.1
- Single-Axis Motion Control, chapter 11.3
- Single-Axis Motion Control in CFC with Visualization-Template, chapter 11.4
- Drive Control via CAM and a Virtual Time Axis, chapter 11.5
- Changing CAMs, chapter 11.6
- Drive Control via the CNC-Editor, chapter 11.7
  - 1: Direct Creation of the Queue, chapter 11.7.1
  - 2: Online Decoding, Use of Variables, chapter 11.7.2
  - 3: Path Preprocessing online, chapter 11.7.3
- Dynamic SoftMotion-Programming, chapter 11.8

11.2 Example: Drive Interface: Create PLC Configuration for Drives

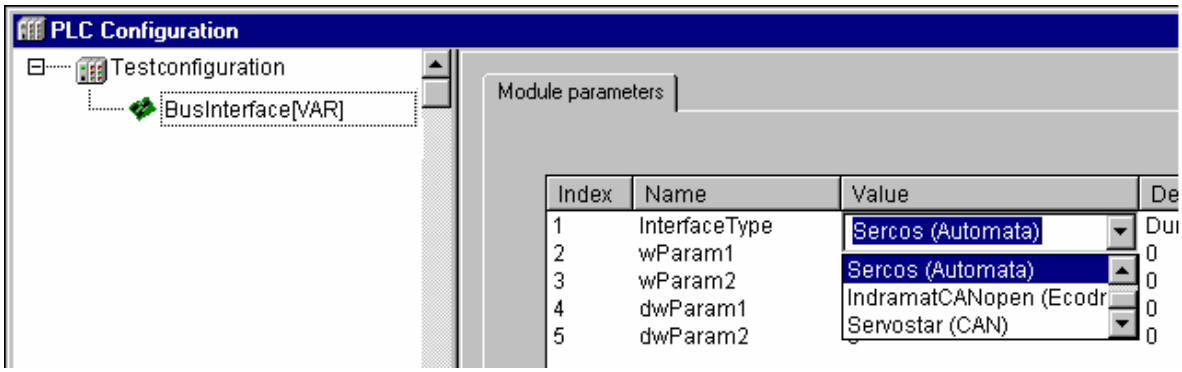
(See the corresponding sample project coming with SoftMotion: DriveInterface.pro, basing on the configuration file softmotion.cfg)

This example describes how to map a given physical drive structure to the IEC program in the CoDeSys programming system. By doing this configuration the IEC program will get access to data structures, which can be used by the SoftMotion modules to create the desired movements.

- First open the **PLC Configuration** (Resources tab) and select 'Extras' 'Standard configuration'.

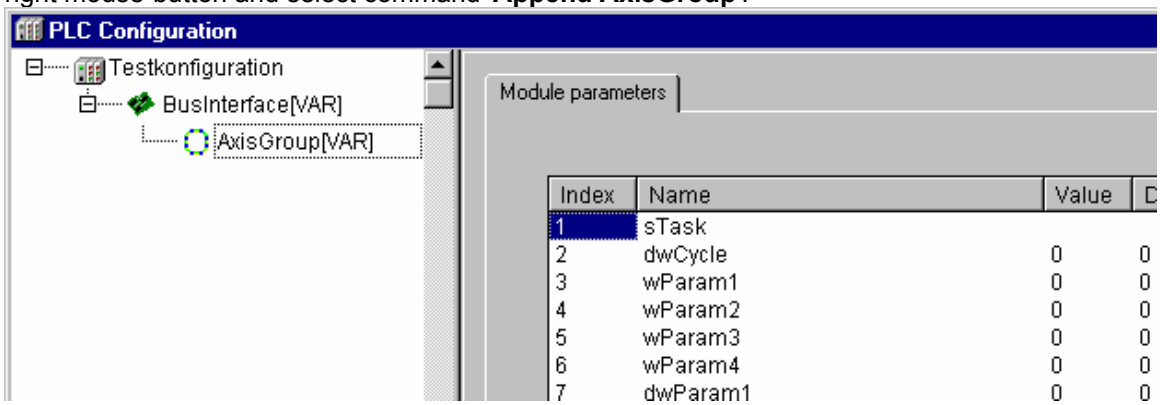
- For this example it is assumed, that a Sercos field bus is used and e.g. an ISA-Bus-Card from Automata. Hence use the command '**Append BusInterface**' in the context menu (right mouse-button) .

- For this bus interface now the appropriate module parameters (Modulparameter) have to be set. First define the "InterfaceType". In our example no hardware is available and we set a kind of simulation mode by selecting "Dummy".



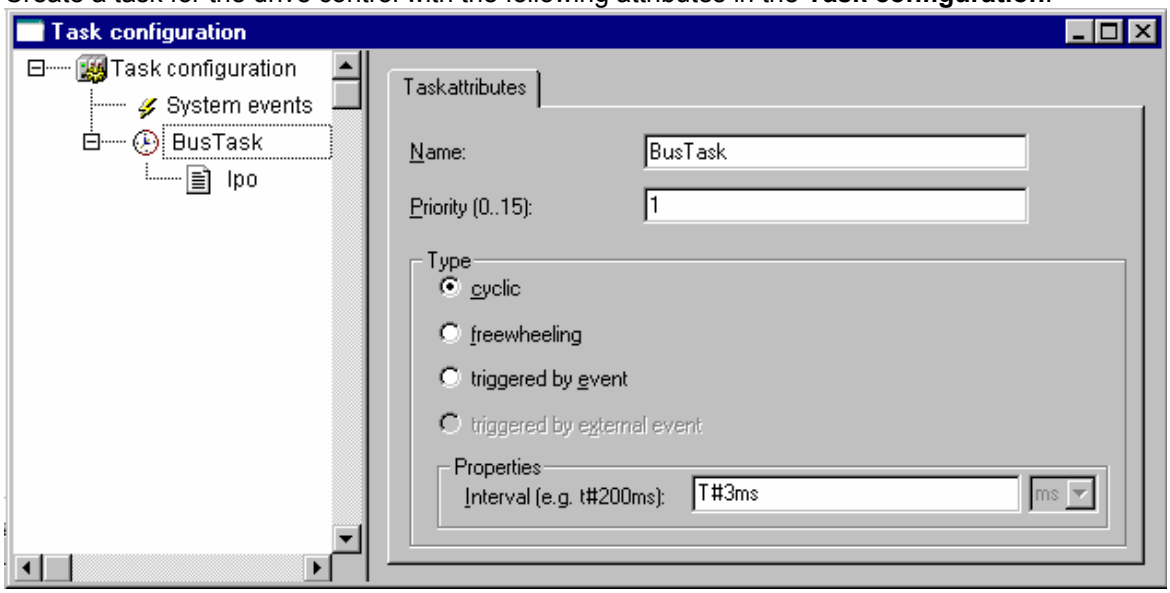
- Depending on the chosen InterfaceType the further parameters have to be defined: E.g. for "Sercos (Automata)" you have to set the interrupt number in wParam1, the hardware type in wParam2 and the base address of the card in dwParam1.

- It is assumed that the card has to serve a ring of four drives. Thus - the entry 'BusInterface' in the configuration tree must be selected (dotted frame) – click in the configuration tree window with the right mouse-button and select command '**Append AxisGroup**':

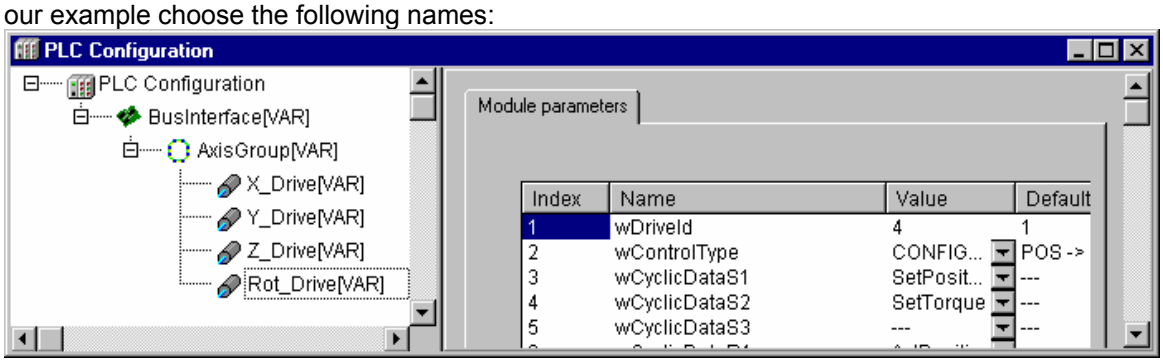


For the AxisGroup now define the **Modulparameter**s. First enter the name of the **Task** (sTask), which will control the drives (e.g. "BusTask"), in the line below enter the cycle time of the task in µsec, e.g. "3000". (See below for the creation of this task.)
The other AxisGroup parameters depend on which BusInterface has been chosen. In our example wParam1 defines the baud rate in MBit (e.g. enter "2") and wParam2 defines the intensity of the LED.

- Create a task for the drive control with the following attributes in the **Task configuration**:



- Now the drives have to be inserted. We assume that there are **four drives**, three linear drives serving a X-, Y-, Z- portal-system, and one drive turning the tool around the Z-axis.
Insert each drive by the command "Append Drive", which is available in the context menu when the entry 'AxisGroup' is selected (dotted line) in the configuration tree. The names of the drives can be modified. For this purpose click on the entry with the right mouse-button to open an edit field. For our example choose the following names:



- Now we are going to parameterize the drives: For this purpose we open the **Modulparameter**-Dialog:

- First define the ID "wDriveId" according to the drive configuration. In our example: "1", "2", "3" and "4". In the following it will be described how the settings in tab „Module parameters“ have to be done; much easier and self-explanatory that also can be done via the dialogs.

- The **portal-drives** each can be moved between -50cm and +50cm. We configure their weighting translational. We use an increment of 10-7 m resp. 10-7m/s for all position and velocity data. Thus all position and velocity data basically have to be evaluated with the unit "10-7m" resp. "10-7m/s". If we want to change this to a unit of "mm/sec", we have to enter "10000" for the parameter *dwRatioTechUnitsDenom* "10000" and "1" for *iRatioTechUnitsNum*. Due to the fact that it is a linear drive, *fPositionPeriod* has no meaning. But you still have to specify the data, which will be sent and received cyclically. For example: If we refer to master telegram, it is sufficient to choose "POS, VEL -> POS, VEL" in the scroll list of *wControlType*. This will cause that cyclically the position and velocity target values will be sent and the current position and velocity values will be received.

  In order to have an additional control of any exceeding of the valid range (-50cm = -5000mm, 50 cm = 5000mm) (the application should be programmed in a way, that this is not possible at all), we activate a control function by setting SWLimitEnable = TRUE, SWLimitNegative = -5000 and SWLimitPositive = 5000.

- We assume a rotation of 65536 increments for the used rotatory drive can be rotated arbitrarily. Thus we define – in order to get an internal unit of angle degrees - "65536" for *dwRatioTechUnitsDenom* and "360" for *iRatioTechUnitsNum*. This drive for example might be designed for turning a screwtop on a bottle. Therefore we want to send cyclically the position and the torque values, because we later want to be able to switch - by a change of the operation mode - from a behaviour which is controlled by the position to a behaviour which is controlled by the torque. To do this settings select "CONFIGURABLE" for w*ControlType* and "fSetPosition" resp. "fSetTorque" for *wCyclicDataS1 resp. wCyclicDataS2*. In order to get returned the current position set the option "fActPosition" for *wCyclicDataR1*.

- In order to get a program ready for an error-free compilation, a **program call has to be appended** to the task "BusTask". For example **create a program** "Ipo", which later will do the motion control, and call it by "BusTask".

Now **compile** the program, no errors should occur, and **load** it to the controller and **start** it. CoDeSys will automatically create the following variables and structures:

→ In the **Global Variables** folder "**Drive Configuration Data**" you will find three instances of the modules "SercosDriveExecute_Start", "SercosDriveExecute_End" and "SercosDriveInit" (elements of the Sercos-library) with the names "AxisGroupStartCycle", "AxisGroupEndCycle" and "AxisGroupInit", which are responsible for the communication with the drives.

→ In the **Global Variables** folder "**Drive_Globale_Variablen**" of the library "**Drive_Basic.lib**" there is a structure variable **g_DRIVESTRUCT**, which contains all entries of the PLC Configuration, i.e. all BusInterfaces, AxisGroups and Drives.

→ Besides that **for each drive globally a structure variable** has been created, e.g. "X_Drive", which can be monitored e.g. in the Watch- and Receipt Manager. This structure can be accessed by the SoftMotion modules and the DriveInterface will keep it up-to-date.

On this drive structures the Motion modules of the IEC program, which we will create in the following in 'Ipo', will work.

11.3 Example: Single Axis Motion Control

(See the corresponding sample project coming with SoftMotion: PLCopenSingle.pro, basing on the configuration file softmotion.cfg

This example shows, how a drive can be controlled via modules conforming to the PLCopen standards:

Besides the **libraries** of the Drive Interfaces the library SM_PLCopen.lib must be included in the project.

In the **PLC Configuration** a linear drive is defined with the name "Drive":

In the **Task configuration** the program "Ipo" is called, which will create a movement on the given axis.

In the following it will be described how to create this **program**:

In the Object Organizer insert a program in Structured Text (ST) and fill it as follows:

Before we program a movement of the drive, we want to make sure that the driver has found and initialized the drive. As soon as this has happened we should unblock the controller and release the brakes if applicable. This is done by the module *MC_Power*:

```
PROGRAM Ipo
VAR
       Init: BOOL := FALSE;
       Power: MC_Power;
END_VAR
IF NOT Init THEN
       Power(Enable:=TRUE, bRegulatorOn:=TRUE, DriveStart:=TRUE, Axis:=Drive);
       Init:= Power.Status;
ELSE
END_IF
```

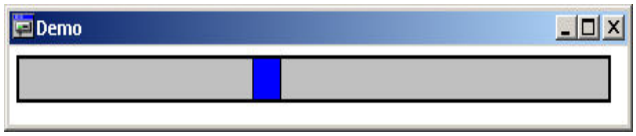
Now the drive can be controlled by the ELSE-part of the first IF-instruction. For our current example we want to do that via the **Positioning Module** MC_MoveAbsolute. For this purpose we define an instance of this module and a target position *p*, which will be initialized with "100". We call this instance in each cycle with the required parameters. As soon as the programmed position has been reached, the *Done* output of the module will be set to TRUE and the *Execute* input must be set to FALSE, if we want to start a new movement, because the module needs a rising edge to start working:

```
...(* Continuation of the above shown program *)

ELSE
       MoveAbsolute(Execute:=TRUE, Position:=p, Velocity:=100,
       Acceleration:=100, Deceleration:=100, Axis:=Drive);
       IF MoveAbsolute.Done THEN
              MoveAbsolute(Execute:=FALSE, Axis:=Drive);
       END_IF
END_IF
```

Now the program can be **compiled** error-free, you can switch to online mode and start the program. Monitoring the current position Drive.fActPosition in a watch list or in the Sampling Trace will show how the drive is moving towards this position. If you force the value of p, the axis will move towards the new target position as soon as the last one has been reached.

For a graphical monitoring of the movement visualization templates for drives are available in library SM_DriveBasic.lib. To use those templates first go offline, create a new visualization and insert a 'visualization' element. From the list, which will show the available visualizations select „LinDrive“. Then perform a double-click on the newly created element and in the dialog 'Visualization' in 'Placeholder...' insert the name of the drive structure (here: "Drive" as a replacement for "AXISREF“. The visualization configured in this way will display the position of the drive:



Example: Single-Axis Motion Control in CFC with Visualization-Template

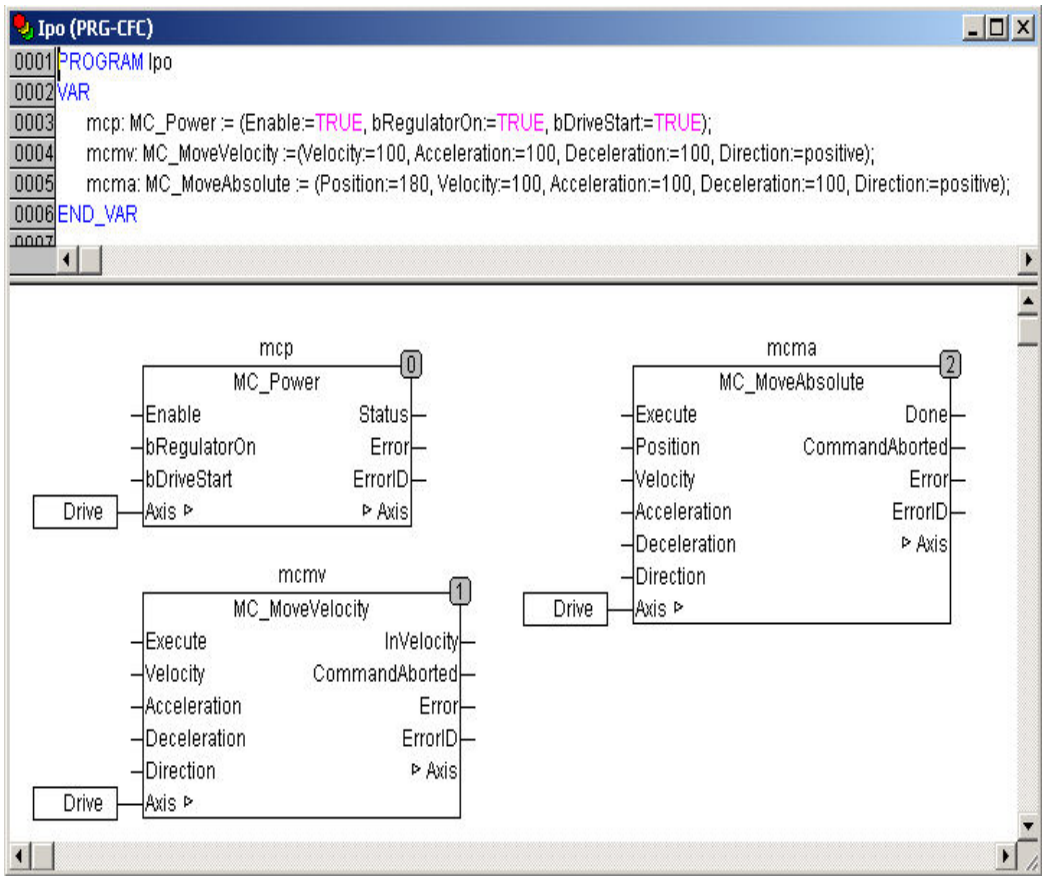
11.4 Example: Single-Axis Motion Control in CFC with Visualization-Template

(See the corresponding sample project coming with SoftMotion: PLCopenSingle2.pro, basing on the configuration file softmotion.cfg)

Like the following example shows, instead of ST you can use any other IEC language for programming.

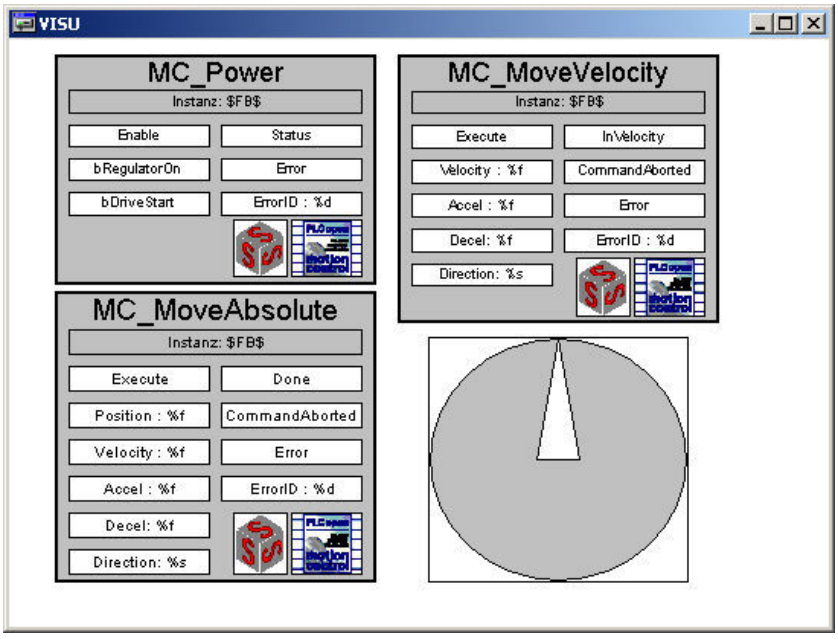
This example should help to understand the start- and interrupt-mechanism of the function blocks in the example project. Additionally the various start modes for module MC_MoveAbsolute for rotatory drives can be tested.

Create a PLC configuration and task configuration like described for the previous example , but this time use a rotatory drive with period 360. Program "Ipo" is written in CFC, it only contains three calls of instances of the function blocks MC_Power (needed for activation of the axis), MC_MoveAbsolute and MC_MoveVelocity:



It is recommended to initialize the inputs of the modules, because then we will not need to re-insert these values each time when we start this test application.

Additionally we create a operation visualization. We use the visualization templates which are available in the libraries and we connect them via the placeholder-concept with the function block instances:



Now we can compile the project without errors, we log in to the controller and start. By pressing the Execute-input of MoveVelocity the drive should start rotating. Press Execute of MoveAbsolute to position the drive to the set position, whereby it will be rotated in positive direction, according to the setting 'Direction: positive'.

This will cause an interrupt of Module MoveVelocity. Play with the modules and test various velocities and accelerations and also test the direction modes (positive/negative/current/shortest/fastest) of MoveAbsolute.

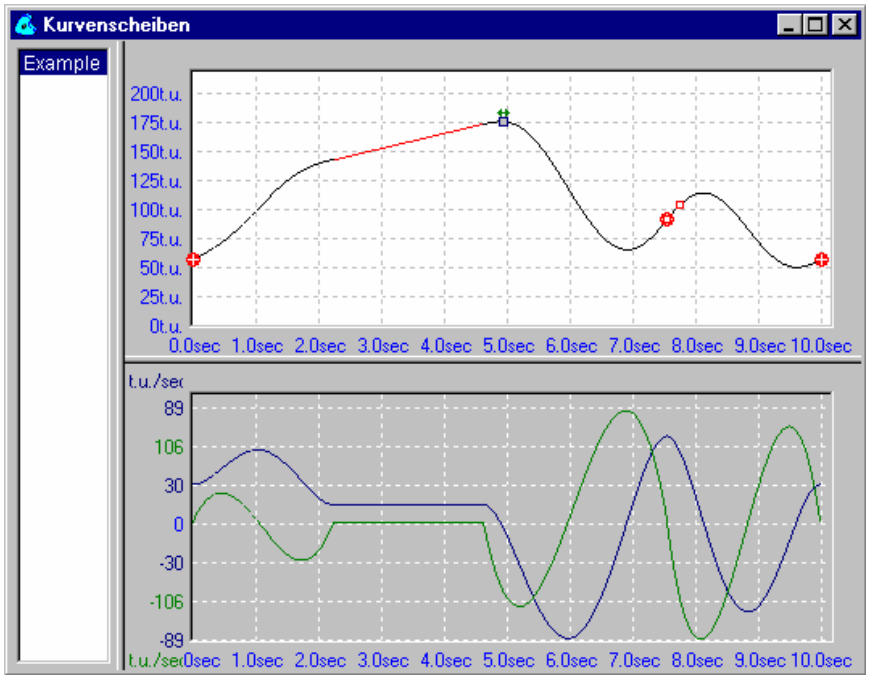
11.5 Drive Control via CAM and a Virtual Time Axis

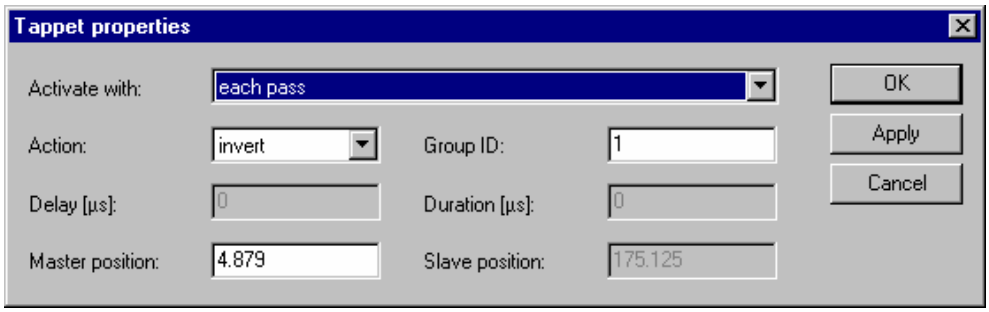
(See the corresponding sample project coming with SoftMotion: PLCopenMulti.pro, basing on the configuration file softmotion.cfg)

(Preconditions: The libraries DriveBasic.lib and SM_PLCopen.lib are included.)

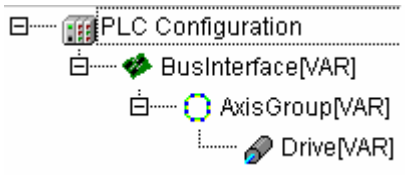
The following example demonstrates how a periodic CAM can be realized on a linear drive. Additionally it shows the tappets function.

1. First create any periodic CAM in the CoDeSys CAM-Editor (Resources tab, CAMs), which refers to a master axis covering a range between 0 and 10 sec and which at least contains one inverting tappet with ID1; for example:





2. In the Drive Interface (PLC Configuration) define a drive "Drive":



3. Create the program 'ipo' in FBD and include the calls of the following modules.

```
PROGRAM Ipo
VAR
Power: MC_Power;
TimeAxis : SMC_TimeAxisFB;
TableSelect: MC_CamTableSelect := (SlaveAbsolute:=TRUE);
amIn: MC_CamIn:=(StartMode:=ramp_in, VelocityDiff:=100,
Acceleration:=100, Deceleration:=100);
Tappet: SMC_GetTappetValue;
END_VAR
```

After the Power-module (MC_Power) for the slave axis first the time axis module will be called (SMC_TimeAxis). Give it a period of 10 seconds, because the CAM is configured for this time . The task cycle time must be inserted manually. TableSelect will select the desired CAM, and CamIn will realize it. The Tappet module checks the position of the tappet. Due to the fact that the tappet is configured 'inverted', it will switch every 10 seconds.

Now you can compile the program and start it on the controller.

In order to control the target resp. current position, create a visualization, which will help to check the paticular modules and the position of the axes.

Regard, that the master of the CAM not only can be a virtual time axis, but of course any desired AXIS_REF data structure. For drives which are currently on regulation, the target values will be regarded, for drives which are not on regulation the current values.

11.6 Example: Changing CAMs

(See the corresponding sample project coming with SoftMotion: PLCopenMultiCAM.pro, basing on the configuration file softmotion.cfg)

(Preconditions: The libraries DriveBasic.lib and SM_PLCopen.lib are included.)

This example shows how a CAM movement with two alternating CAMs can be realized. It has been programmed in ST and performs the same actions like shown in the preceeding example. At the end of the first CAM the MC_CamIn module sets the output bEndOfProfile, which will cause that the currently other MC_CamTableSelect will be used and restarted together with MC_CamIn.

11.7 Example: Drive Control via the CNC-Editor

In **three parts** this CNC Example will show the basic structure of a possible CoDeSys IEC program, which can realize the pathes designed in the CNC-Editor.

Like described there are two possibilities to compile and to use a CNC program.

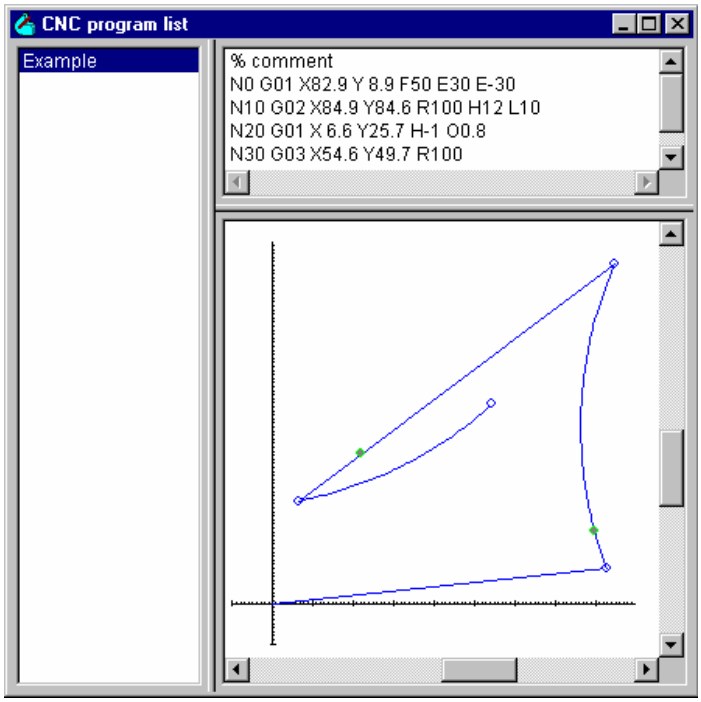
The first part of the example shows the direct creation of an OutQueue, the second part shows the online decoding of the program by using variables. The third part of the example shows how additionally to use a path-preprocessing module.

11.7.1 CNC Example 1: Direct Creation of the OutQueue

(See the corresponding sample project coming with SoftMotion: CNCdirect.pro, basing on the configuration file softmotion.cfg)

1. Creation of the NC program in the CNC-Editor:

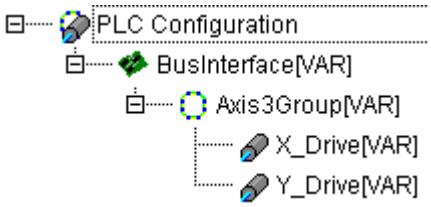
We create an example program, which is running between x out of [0,100] and y out of [0,100]. Additionally we define the velocities and accelerations for the path and set two witch points on the path. E.g.:



As compile mode we choose „create OutQueue on compile“.

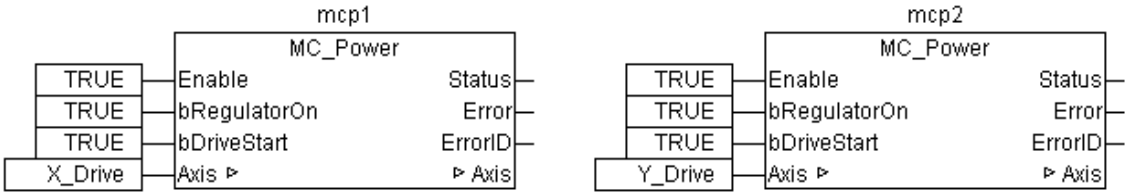
2. Drive Interface, PLC configuration:

Define a drive structure with 2 linear drives; the maximum velocity etc. is to be set.

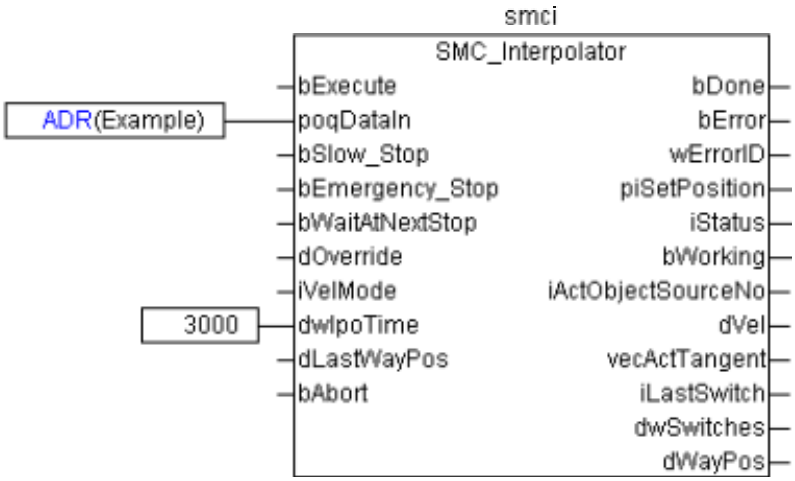


3. Creation of the IEC program:

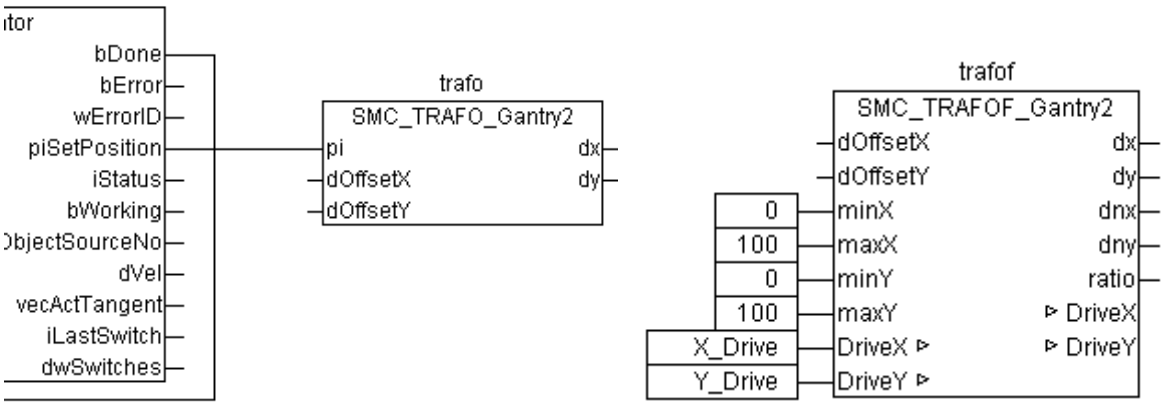
First we have to activate the drives via the MC_Power module:



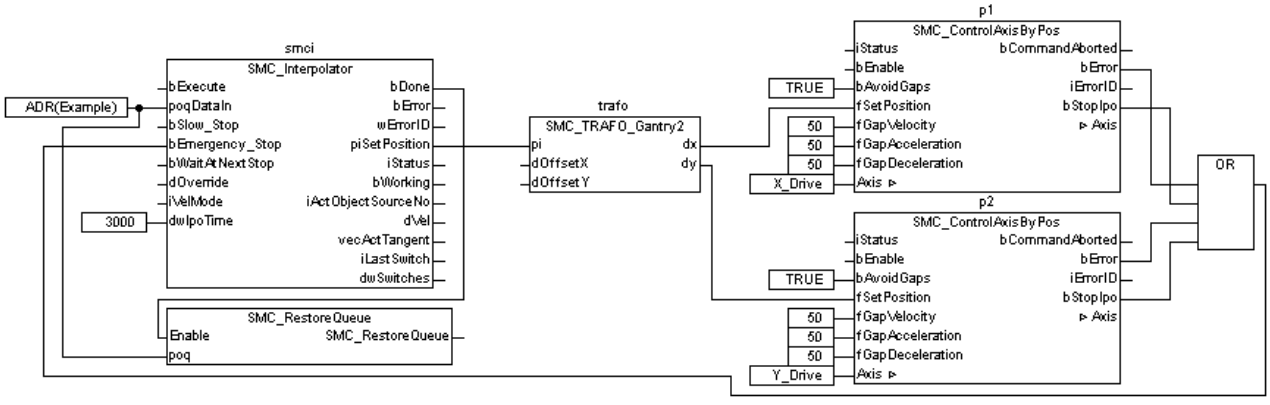
A further important element is the SMC_Interpolator module. As input poqDataIn it gets the address of the CNC program. Besides that the IEC- task cycle time must be written to dwIpoTime.



We want to control a portal system with this example program. For this reason we insert an instance of the backward and forward transformation modules from library SM_Trafo.lib. The forward transformation module as inputs gets the drives (the Z-drive is allocated with a otherwise not used variable dummy of type AXIS_REF); the backward module must get the target position of the Interpolator:



The outputs of the module, that means the axes coordinates, now must be written to the drives. For that the function blocks SMC_ControlAxisByPos are used. Due to the fact that our application does not guarantee continuous outputs of the Interpolator (e.g. the path ends at a point different to that where it starts), we should activate gap avoiding functions (bAvoidGaps, fGapVelocity, fGapAcceleration, fGapDeceleration), we should connect the StopIpo-output with the bEmergency_Stop of the Interpolator and we should connect the Interpolator-output iStatus with the corresponding inputs of the axes control modules.



Please regard during programming in CFC the correct order of the elements !

4. Creating the operation and test interface:

Create a new visualization and insert two visualization elements of type 'Visualization'. The first one is the Interpolator template, the second one is the Transformation template. They get linked to the corresponding function block instances (here: Ipo.smci resp. Ipo.trafof) via the placeholder functionality.

5. Starting

The program now can get compiled without errors and can be started. It will execute the CNC program, as soon as the Execute-input of the Interpolators gets set. After having been processed completely, an other rising edge will cause a new run.

Regard the function of the path switches, which are also displayed in the visualization of the Interpolator module.

11.7.2 CNC Example 2: Decoding online with use of variables

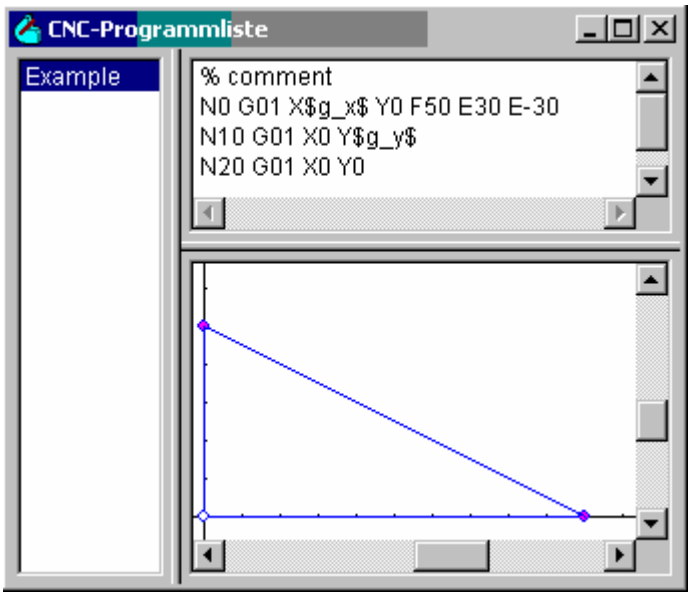
(See the corresponding sample project coming with SoftMotion: CNConline.pro)

1. Creating the NC program in the CNC-Editor:

Like in the previous example we create a CNC program, but now we use two global variables g_x and g_y. E.g.:

```
VAR_GLOBAL
    g_x: REAL:=100;
    g_y:REAL:=50;
END_VAR
```

We choose compilation mode 'Create program variable on compile' because we are using variables in our CNC program.

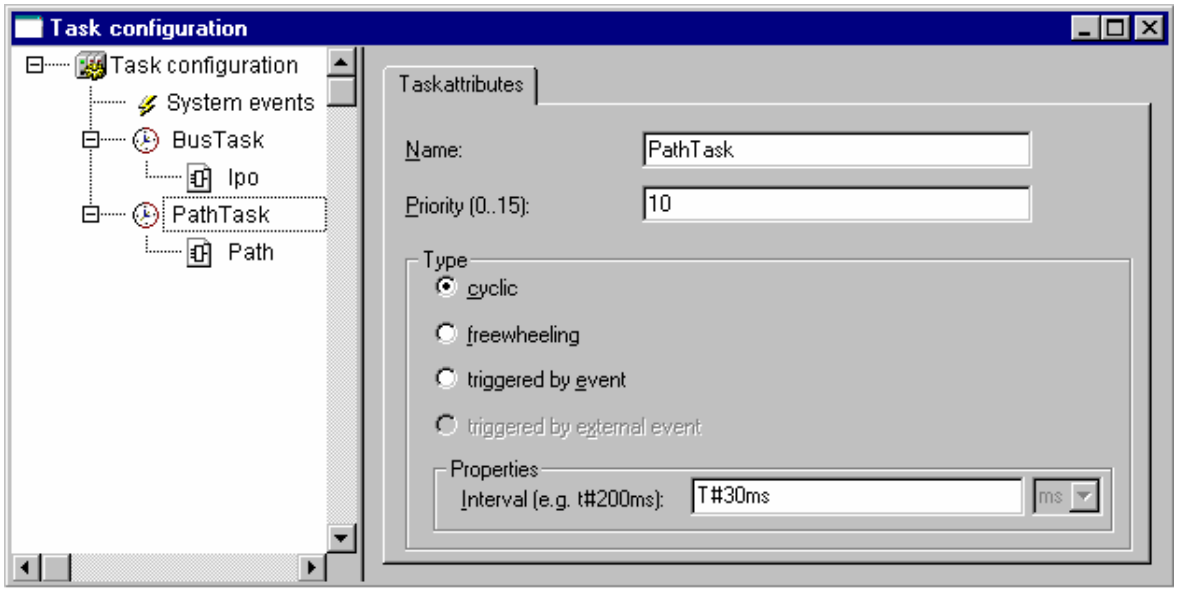


2. Drive Interface, PLC configuration:

The drive structure is the same as described in Example 1.

3. Creating the IEC program:

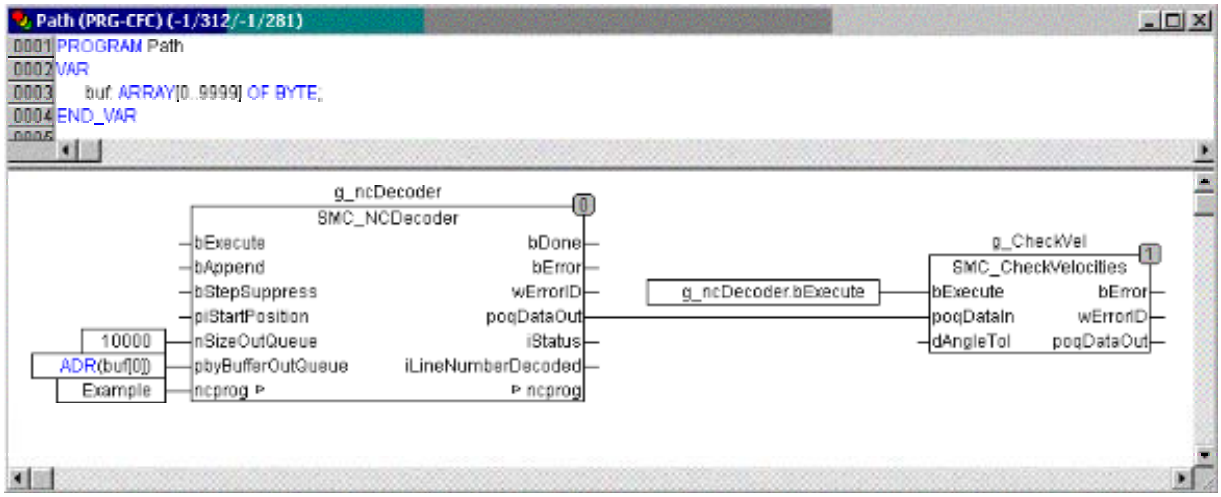
Due to the fact that we have chosen a different compile mode for the current example, we have to do the decoding and path-preprocessing in the IEC program. This time-consuming process must not be done in time with the Interpolator (reason: at each call of the Decoder a path object is created, which typically can be used for many Interpolator calls), thus the Interpolator often is swapped out to a task of lower priority, which is called more rarely:



The underlying mechanism: In the slow task initially about one GEOINFO-object will be created per cycle, which is stored in the OUTQUEUE-structure of the Decoder module.

As soon as this OUTQUEUE is filled, the modules of the slow task will pause until the OUTQUEUE isn't full any longer, this means until the fast task has processed the first GEOINFO-object and removed it from the OUTQUEUE. In this case the modules of the slow task get active again and re-fill the OUTQUEUE-structure. In the fast task per each cycle one path position point of that OUTQUEUE-structure, to which the DataIn parameter of the Interpolator is currently pointing, will be calculated and processed. Due to the fact that a GEOINFO-object generally consists of several position points, it will take several cycles until the first GEOINFO-object has been processed completely and will be removed by the Interpolator. Since the processing of a GEOINFO-object obviously takes more cycles than the creating, the slow task actually can be called more rarely than the fast. However, the task times must be defined in a way which makes sure that in the last OUTQUEUE of the slow task always enough GEOINFO-objects are available, so that no data-underrun can occur. A data-underrun would result if the Interpolator does not get any GEOINFO-objects from DataIn while the end of the path has not yet been reached.

In program Path the decoding of the NC program and the velocity check are done:



The interpolating part of the IEC program nearly remains at it was, except that the data input of the Interpolator is not the CNC program name (ADR(Example)), but the OutQueue-output of the path-preprocessing modules (here: g_ncDecoder.poqDataOut).

Besides that function SMC_RestoreQueue should not be called.

4. Creating the operation- and test interface:

For a visualization of the previous example it is useful to add templates of the new modules (SMC_NCDecoder and SMC_CheckVelocities). Besides that the global variables g_x and g_y should be editable, in order to be able to check their function later at start-up.

5. Start-up:

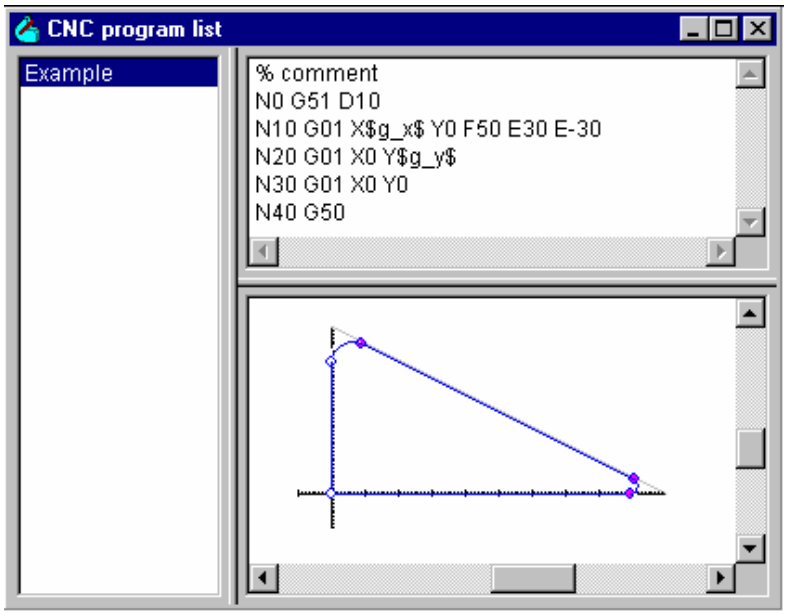
The program now can be compiled without errors and started. It will process the CNC program as soon as the Execute-inputs of the Decoder and Interpolator have been set.

If you change the values of the global variables, those will be read at a re-start of the Decoder and the path will be adjusted accordingly. Also regard the function of the Append-input of the Decoder.

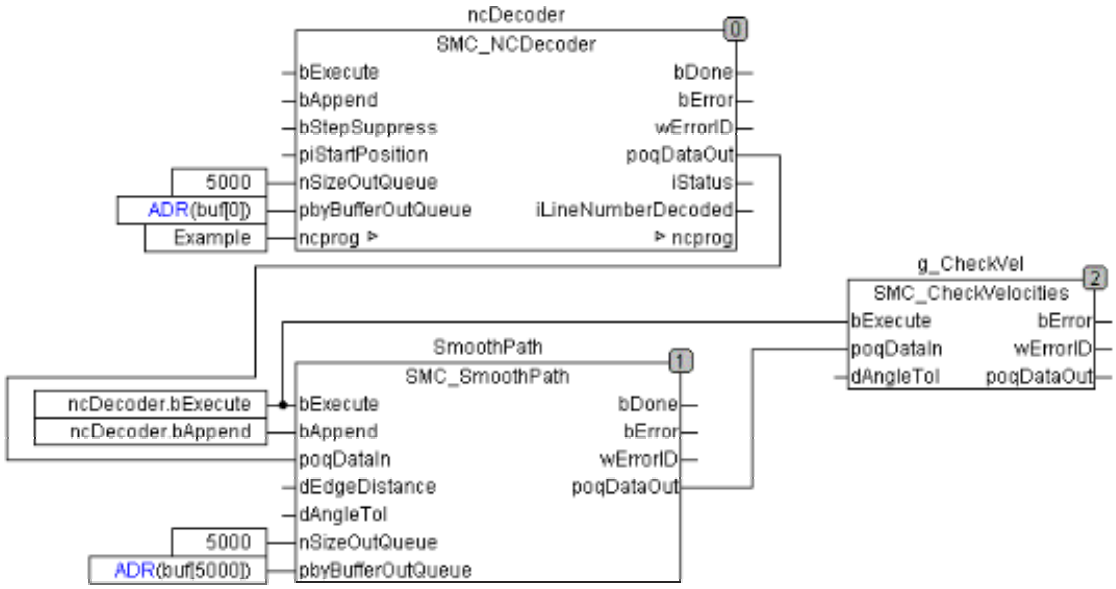
11.7.3 CNC Example 3: Path-Preprocessing online

(See the corresponding sample project coming with SoftMotion: CNCprepro.pro)

We want to extend the previous example by a path-preprocessing function: The corners of the program should be smoothed by splines. This is done by the SMC_SmoothPath module. The CNC program must be embraced by the words G51/G50:



If we did not use variables, we could compile the program as it is as a Queue and could directly pass it on to the Interpolator. But using variables, we must do the decoding and rounding off in the program. For this purpose we define a new POU of type SMC_SmoothPath and call it after the Decoder:



The data input of the Interpolator module as always must be connected to the output poqDataOut of the CheckVelocities module.

This program can get compiled without errors and will not stop - contrary to the previous one - in the corners of the NC program, because the corners now have been smoothed by the path-preprocessing module.

11.8 Dynamic SoftMotion-Programming

(See the corresponding sample project coming with SoftMotion: CNCDynamicPath.pro, basing on the configuration file softmotion.cfg)

One of the advantages of SoftMotion is, that the programmer and the user not only can influence the processing of a path but also can create and modulate this path while the program is running. In order to reach this the programmer just has to replace the Decoder module by an self-designed path generator. Nevertheless the path preprocessing and especially the Interpolator can be used further on as usual.

In order to replace the Decoder module, the OUTQUEUE-structure object must be created in another way. It must also be filled with GEOINFO-objects, which represent the desired path, and it must be passed on to the appropriate subsequent module (e.g. Interpolator).

Preparing steps:

- In the declaration part an OUTQUEUE-, a GEOINFO-structure object and a buffer of desired size must be defined:

```
QUEUE: SMC_OUTQUEUE:=(nSize:=10000);
BUF: ARRAY[0..9999] OF BYTE;
GEO: SMC_GEOINFO:=(dT1:=0, dT2:=1, dToolRadius:=0, dVel:=100, dVel_End:=100,
Accel:=200, dDecel:=500, iObj_Nr:=0);
```

- In an Init-Step in the implementation part of the program the OUTQUEUE-structure must get initialized:

```
QUEUE.nSize := 10000;
QUEUE.pbyBuffer := ADR(BUF[1]);
```

Dynamc Path-Programming

In the program body, there where you want to create the path, implement the following steps for each GEOINFO-object:

- Set start-position (first object)

```
GEO.piStartPos.dX := 0;
```

...

...resp. copy from the preceeding object.

```
GEO.piStartPos := GEO.piDestPos;
```

- Define move-type. e.g.:

```
GEO.iMoveType := CCWL;
```

or

```
GEO.iMoveType := LIN;
```

- Set the parameters required by the chosen move-type. If you have defined a circular arc (e.g. CCWL), do not forget to set the following parameters (see structure SMC_GEOINFO):

```
GEO.dP1 := 200;
GEO.dP2 := 100;
GEO.dP3 := 50;
GEO.dT1 := 0;
GEO.dT2 := 90;
```

- if applicable, set the start- or end-bit in InternMark for the path preprocessing (see structure SMC_GEOINFO).

- Calculate the end-position:
  `SMC_CalcEndPnt(ADR(GEO));`

- Calculate the length of the object:
  `SMC_CalcLengthGeo(ADR(GEO));`

- Store the object to OUTQUEUE:
  `SMC_AppendObj(POQ:=ADR(QUEUE), PGI:=ADR(GEO));`

- As soon as the path has been created completely, the OUTQUEUE-list must be closed:
  `QUEUE.bEndOfList := TRUE;`

Regard, that, if the OUTQUEUE is full, i.e. if QUEUE.bFULL = TRUE, the program will not try any longer to add further objects. In this case the creation of the path must be interrupted until the first object of the OUTQUEUE has been processed. Then a further object can get appended. If you want to avoid this case, you must define the size of OUTQUEUE sufficiently high so that all GEOINFO-objects of the desired path can be catched by it.

The object list queue finally will be passed on first to the CheckVelocities module and finally to the Interpolator, which will process it further on.

In this example you also see how a kinematic transformation, which is not provided by the 3S library SM_TRAFO.lib, can be programmed manually. The modules SMC_TRAFO and SMC_TRAFOF, which are included in the project, show this for the example of a cartesian X/Y-system.

12 Index

A

angle value 2-7
Avoid loop 3-11
AXIS_REF structure 2-16
axisgroup 2-8
Axisgroup 2-1
AxisGroup 2-2

B

BusInterface 2-1, 2-2

C

CAM 4-1
  Edit mode 4-3
  editing 4-4
  Element properties 4-4
  master axis 4-1
  periodic 4-2
  Properties 4-2
  slave axis 4-1
CAM definition for SoftMotion 4-1
CAM disc
  master axis 4-1
CAM Element Properties 4-6
CAM Function Blocks 10-3
CAM_REF 4-8
CAM-Editor 1-2, 4-1
  Create new CAM 4-2
  Definition of a CAM for SoftMotion 4-1
  Insert
    Select elements 4-7
  Insert line 4-8
  Insert point 4-8
  Insert tappet 4-8
  Start 4-1
CAMXYVA 4-9
CAN drive settings 2-7
CAN specific settings 2-2
Circle CCW Insert Mode 3-10
Circle CW Insert Mode 3-9
CNC language 3-2
CNC program 3-4
  line number 3-3
  sentence 3-2
  sentence number 3-2
  word 3-2, 3-3
  word identifier 3-2
CNC program Menu in the CNC-Editor 3-4
CNC-Editor 1-2, 3-1, 3-4
  create program 3-4
  Define queue size 3-6
  Define start position 3-6
  Delete 3-5
  Divide object 3-7
  Graphic Editor 3-8
  Info 3-5
  Invert direction 3-6
  Move program 3-6
  Rename CNC Program 3-5
  Rotate program 3-6
  Split object 3-7
  Start 3-4
  Stretch program 3-6
  Text editor 3-8
Code Generation 2-7
Compile options 4-6
ControlAxis function blocks 2-10
Convert splines/ellipses to lines 3-10
Cycletime for axisgroup 2-2

D

Define queue size 3-6
Define start position 3-6
delete CNC program 3-5
Diagnosis 2-14
Divide object 3-7
don't compile 4-7
Drive 2-1, 2-2, 2-5
Drive dialog 2-4
Drive id 2-4
Drive Interface Sample Configuration 11-1
Drive parameters 2-20
Drive.lib 2-15
DriveInterface 1-1
Driver 2-15

E

Editing Modes in the CNC-Editor 3-8
element optimized point table 4-7
Ellipses 3-10
Encoder 2-15
Encoder 2-2
Encoder settings 2-7
equidistant point table 4-7
Error handling 9-1
Error numbers 9-1
Export CAM as ASCII-table 4-7
Extras
  Avoid loop 3-11
  Circle CCW Insert Mode 3-10
  Circle CW Insert Mode 3-9
  Compile options 4-6
  Convert splines/ellipses to lines 3-10
  Export CAM as ASCII-table 4-7
  Fit to Screen 3-10
  Line Insert Mode 3-9
  Read CAM from file 4-7
  Renumber program 3-8, 3-10
  Round off path 3-11
  Select Mode 3-9
  Set epsilon values 3-11
  Settings 4-4, 4-6
  Show bounds 4-6
  Show complete CAM 4-4, 4-6
  Show grid 3-10
  Show Interpolation Points 3-11
  Slur path 3-10
  Spline Insert Mode 3-10
  Step Suppress 3-11
  Tool radius correction 3-10
  Write CAM to file 4-7
Extras Menu in the CAM-Editor 4-6

F

File for CNC program 3-1
Fit to Screen 3-10

G

GantryCutter 8-3
global Variables in SM_CNC lib 6-17
Graphic Editor 3-8

I

Info on CNC Program 3-5
Insert
  Insert line in the CAM-Editor 4-8
  Insert point in the CAM-Editor 4-8
  Insert tappet in the CAM-Editor 4-8
  New CAM 4-2
  Properties 4-2
  Select elements in the CAM-Editor 4-7
Insert line 4-8
Insert Menu in the CAM-Editor 4-6
Insert Mode in the CNC-Editor 3-8
Insert point 4-8
Insert tappet 4-8
Interpolation 6-14
Invert direction 3-6

J

Jerk 2-4

L

Library
  Drive_Basic.lib 2-7
  Manufacturer specific Drive-Lib 2-7
  SM_CNC.lib 6-1
  SM_CNCDiagnostic.lib 7-1
  SM_Error.lib 9-1
  SM_PLCopen.lib 5-1
LinDrive 2-15
LinDrive_V 2-15
Line Insert Mode in the CNC-Editor 3-9
line number 3-3
Linear drive 2-15
Loop 6-5

M

master axis 4-1
Mathematic modules 2-7
MC_AbortTrigger 5-15
MC_AccelerationProfile 5-13
MC_CamIn 5-16
MC_CamOut 5-17
MC_CamTableSelect 5-15
MC_GearIn 5-18
MC_GearOut 5-18
MC_Home 5-5
MC_MoveAbsolute 5-6
MC_MoveAdditive 5-7
MC_MoveRelative 5-8
MC_MoveSuperImposed 5-9
MC_MoveVelocity 5-10
MC_Phasing 5-18
MC_PositionProfile 5-11
MC_Power 5-4
MC_ReadActualPosition 5-4
MC_ReadActualTorque 5-4
MC_ReadActualVelocity 5-4
MC_ReadAxisError 5-2
MC_ReadBoolParameter 5-3
MC_ReadParameter 5-3
MC_ReadStatus 5-2
MC_SetPosition 5-14
MC_Stop 5-5
MC_TouchProbe 5-14
MC_VelocityProfile 5-12
MC_WriteBoolParameter 5-3
MC_WriteParameter 5-3
Menu CNC program
  Define queue size 3-6
  Define start position 3-6
  Delete 3-5
  Divide object 3-7
  Info 3-5
  Invert direction 3-6
  Move program 3-6
  New CNC program 3-4
  Rename CNC Program 3-5
  Rotate program 3-6
  Split object 3-7
  Stretch program 3-6
  Write outqueue in file 3-7
Module parameters 2-5
modulo 2-7
Modulparameter 2-1
Motion 1-1
Motion task 2-2
Move program 3-6

N

New CNC program 3-4

O

OUTQUEUE 6-17
override 6-8, 6-11

P

path object 6-17
path section 6-17
Path-preprocessing 6-6
periodic CAM 4-2
PLC Configuration for drives - Example 11-1
PLC Configuration for SoftMotion 2-1
PLCopen.lib 1-2
Point 4-4
polynomial compilation 4-7
Portal systems 8-3, 8-8
Portal Systems\with Tool Offset 8-4
Position data 6-17
Position saving 6-17
Properties of a CAM 4-2

R

Read CAM from file 4-7
Reference move 2-12
Rename CNC Program 3-5
Rotate program 3-6
Rotation of a path 6-16
Rotatory drive 2-15
RotDrive 2-15
Round off path 3-11
Round path 6-8
running order 3-2

S

Scara-Systems 8-9, 8-11
Select elements in the CAM-Editor 4-7
Select Mode in the CNC-Editor 3-8, 3-9
Sercos drive settings 2-6
Sercos settings 2-2
SercosDrive.lib 2-15
Set epsilon values 3-11
Settings in the CAM-Editor 4-6
Settings of a CAM 4-2
Shifted path 6-3
Show bounds 4-6
Show complete CAM 4-6
Show Interpolation Points 3-11
sign value 2-7
Single-axis motion control 11-4
slave axis 4-1
slur path 6-6
Slur path 3-10
SM_CAN.lib 2-16
SM_CNC libraries 1-2
SM_CNC.lib 1-2, 6-1
  OUTQUEUE 6-20
SM_CNCDiagnostic.lib 1-2, 7-1
SM_Error.lib 9-1
SM_Error.lib 1-2
SM_FileFBs.lib 1-2
SM_PLCopen.lib 1-1, 5-1
  MC_PositionProfile 5-11
SM_Trafo.lib 1-2
SMC sgn 2-7
SMC_atan2 2-7
SMC_AvoidLoop 6-5
SMC_CalcDirectionFromVector 8-4
SMC_CAMEditor 5-19
SMC_CAMRegister 5-20
SMC_CAMTable_<variables-type>_<number of elements>_1 4-10
SMC_CAMTable_<variable-type>_<number of elements>_2 4-10
SMC_CAMVisu 5-19
SMC_CMC_REF 6-22
SMC_CNC_REF-data 7-1
SMC_ControlAxisByPos 2-10
SMC_ControlAxisByPosVel 2-11
SMC_ControlAxisByVel 2-11
SMC_CoordinateTransformation3D 8-15
SMC_DetermineCuboidBearing 8-16
SMC_Error 9-1
SMC_ErrorString 9-1
SMC_fmod 2-7
SMC_GCode_Word 6-23
SMC_GEOINFO 6-17
SMC_GetAxisGroupState 2-8
SMC_GetCamSlaveSetPosition 5-19
SMC_GetMaxSetAccDec 2-14
SMC_GetMaxSetVelocity 2-14
SMC_GetTappetValue 5-21
SMC_GetTrackingError 2-14
SMC_Homing 2-12
SMC_Interpolator 6-11
SMC_Interpolator2Dir 6-15
SMC_Interpolator2Dir_SlowTask 6-15
SMC_NCDECODER 6-1
SMC_OutQueue 6-20
SMC_OutQueue-data 7-1
SMC_POSINFO 6-17
SMC_ReadCAM 10-3
SMC_ReadNCQueue 10-1
SMC_ReadSetPosition 5-22
SMC_Reset 5-2
SMC_ROTATEQUEUE2D_2D 6-16
SMC_RoundPath 6-8
SMC_SetTorque 5-22
SMC_ShowCNCREF 7-1
SMC_ShowQueue 7-1
SMC_SmoothPath 6-6
SMC_TimeAxisFB 2-12
SMC_ToolCorr 6-3
SMC_TRAFO_Gantry2 8-3
SMC_TRAFO_Gantry2 8-8
SMC_TRAFO_Gantry2Tool1 8-5
SMC_TRAFO_Gantry2Tool2 8-6
SMC_TRAFO_Gantry3 8-2
SMC_TRAFO_Scara2 8-10
SMC_TRAFO_Scara3 8-11
SMC_TRAFOF_Gantry2 8-2
SMC_TRAFOF_Gantry2 8-9
SMC_TRAFOF_Gantry2Tool1 8-5
SMC_TRAFOF_Gantry2Tool2 8-7
SMC_TRAFOF_Gantry3 8-3
SMC_TRAFOF_Scara2 8-10
SMC_TRAFOF_Scara3 8-12
SMC_TRAFOV_Gantry 8-3
SMC_TRANSLATEQUEUE3D_2D 6-16
SMC_UnitVectorToRPY 8-16
SMC_VARLIST 10-2
SMC_VECTOR3D 6-17, 6-20
SMC_VECTOR6D 6-20
SMC_WriteDriveParamsToFile 2-21
smooth path 6-6
SNC_ChangeGearingRatio 2-9
SNC_ISAxisGroupReady 2-8
SNC_ResetAxisGroup 2-8
SNC_SetControllerMode 2-10
SoftMotion CNC-Library 6-1
SoftMotion Drive Interface 2-1
SoftMotion_CNC_Globals 6-17
Spacial Transformation 8-15
specific settings for CAN axisgroup 2-2
Spline Insert Mode 3-10
Splines 3-10
Split object 3-7
standard language 3-2
Step Suppress 3-2, 3-11
Stretch program 3-6

T

Tappet 4-4

Time axis 2-12
Tool radius correction 3-10, 6-3
Trafo.lib 8-3
Transformation
  spacial 8-15
Transformation function blocks 8-3
Translation of a path 6-16

V

Velocity ramp mode 2-4
Virtual time axis 2-12
Visualization templates 2-15

W

word identifier 3-2
Write CAM to file 4-7
Write outqueue to file 3-7